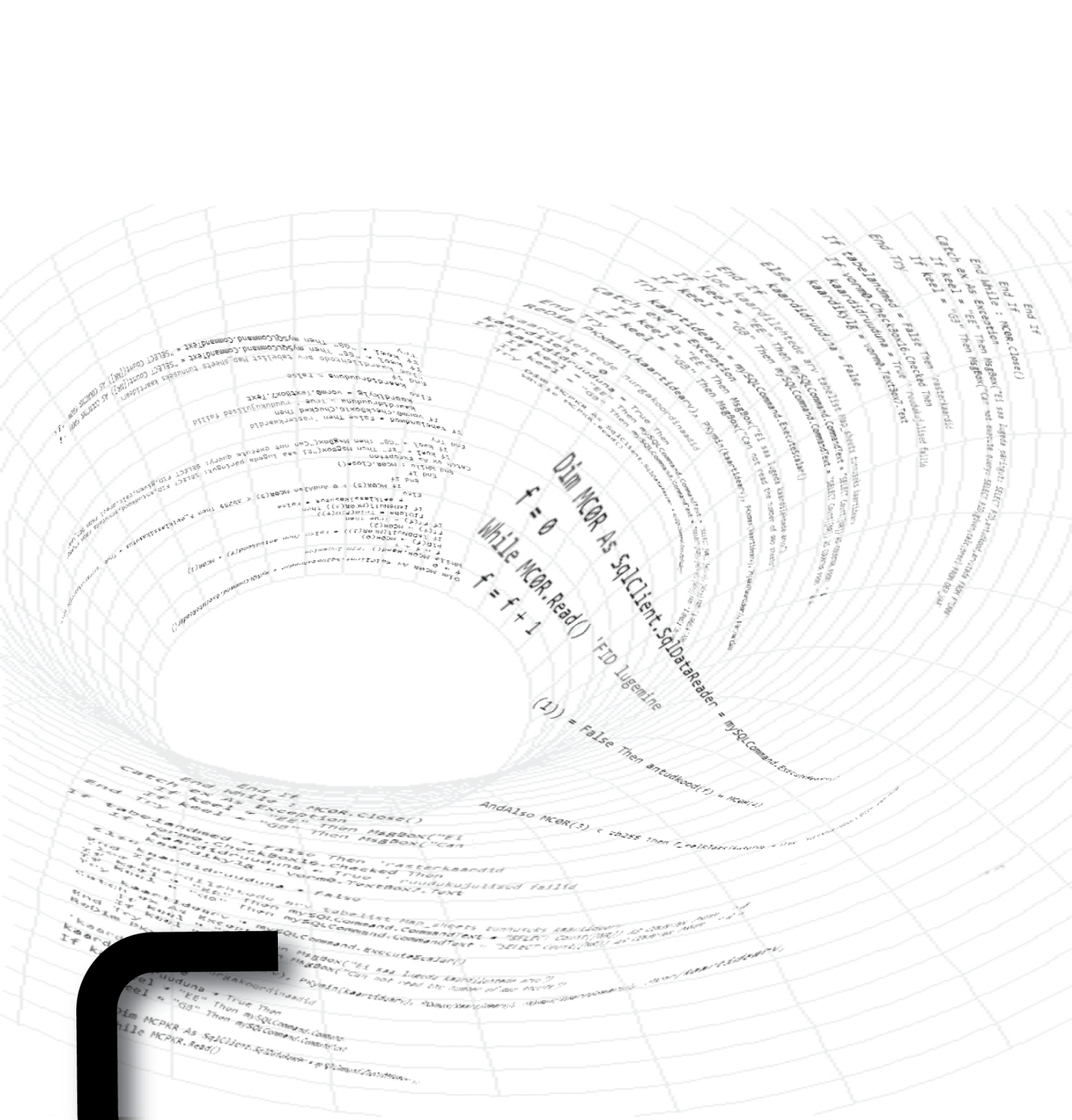

Dim MCOR As SqlClient.SqlDataReader = mySQLCommand.ExecuteReader
f = 0
While MCOR.Read() 'FID lugemine
f = f + 1

CONSTUD

Constud Tutorial

Published in DSpace digital repository, University of Tartu Library

http://dspace.utlib.ee/dspace

Proofreader: Dirk Jonathon Lloyd

Reviewers:
Haris Kontoes (Institute for Space Applications and Remote Sensing National Observatory of Athens Metaxa & Vas. Pavlou),
Tõnu Möls (Assistant Professor emeritus, Faculty of Mathematics and Computer Sciences, University of Tartu; Institute of Agricultural and Environmental Sciences, Estonial University of Life Sciences)

Constud web site: http://www.geo.ut.ee/CONSTUD

Constud version May 2011 source code author: Kalle Remm.

Constud version May 2011 source code property owner: University of Tartu.

The publication was supported by the Departement of Geography, University of Tartu.

ISBN: 978-9985-4-0679-3 (pdf)

Constud Tutorial

Kalle Remm and Tiiu Kelviste

Chair of Geoinformatics and Cartography, University of Tartu
Tartu 2011

CONTENTS

Preface

Constud is a software system designed and coded by Kalle Remm in 2002–2011. Madli Linder, Hendrik Proosa, Tanel Tamm, Tiiu Kelviste, Eerik Absalon, Joonas Remm and Allan Rajavee participated in system testing and writing technical documentation. Constud's source code is property of the University of Tartu.

The **Constud** description and application examples were published by *Remm (2004)* (basics of the machine learning and prediction algorithm used in **Constud**), *Remm and Remm (2008)* (a general description of the system and application on non-spatial data), *Remm and Remm (2009)* (a description and application of the system for predictive mapping of species' distribution), and *Remm and Remm (2010)* (comparison with data mining methods).

The authors would like to express their gratitude to Prof. Ülo Mander for finding financial support, to the aforementioned assistants, to the students who studied *Remote Sensing 2* during spring semester in 2011, and to dear colleagues who motivated the writing of this manual with their claims that it takes at least five years to learn and understand **Constud**. Hopefully this manual helps users to manage within a shorter time. Five days should be enough to become an average user.

The authors express their gratitude to Dirk Jonathon Lloyd for proofreading the text.

The authors

1 INTRODUCTION

Constud (**Con**tinuous **stud**y) is a MS Windows-based software system designed for:

1) calculating indices characterizing patterns in raster format data layers to a database table or to a binary raster file;

2) machine learning — iterative search and weighting of features and exemplars needed for the most reliable similarity-based predictions;

3) calculating similarity-based predictions of nominal and numerical variables to a database table or to a binary raster file.

Constud software is tested in Windows XP, Windows Vista and Windows 7.

Predictions are commonly calculated from models or deduced as expert decisions. **Constud** is a software system for similarity-based or case-based reasoning. This means the predicted value is derived from the values of the most similar exemplars. Similarity-based reasoning is simpler than model-based reasoning because it uses raw data instead of theoretical models. The only rules applied in similarity-based reasoning are the similarity function and the rules for finding the most similar exemplars from the existing knowledge base or from learning observations. The user of a case-based system does not involve his/her subjective understanding about the nature of the relationships by preselecting a model.

The main advantages of the **Constud** system are found firstly in its similarity-based reasoning that enables one to use different types of dependent and explanatory variables, and secondly in its integration of the main stages of predictive mapping into one software environment: 1) the calculation of spatial features, 2) fitting the predictive set of features and exemplars to learning data, 3) calculation of predictive maps. **Constud** does not involve calculation of any probability and statistical significance, it is designed mainly for prediction tasks, incl predictive mapping.

Potential users of **Constud** include those who are interested in data mining and pattern recognition methods, and/or need software for calculating predictive maps. **Constud** is reasonably universal but has hitherto not been user friendly. The authors welcome new users while exercising caution in suggesting **Constud** to less-dedicated users because of the complex and strictly fixed database structure which pertains to **Constud**.

The Constud system enables to use different types of dependent and explanatory variables. Constud integrates: 1) the calculation of spatial features, 2) fitting the predictive set, and 3) predictions to maps and to tables.

The system consists of three parts:

1) **data layers** of explanatory variables (features), pre-classifiers and interpolation polygons (only when spatial data are used);

2) **a database** of observations (cases), parameters of machine learning, results of machine learning and predicted values;

3) **the software** application **Constud**.

The 3 components of the Constud system are: data layers, the database, and the software application

Constud functions only in conjunction with a strictly fixed database structure. Besides the database, **Constud** interacts with uncompressed, unpacked, binary raster files without headers (**Idrisi32** rst-format). Vector format data layers must be rasterised; other raster formats must be converted to **Idrisi**. The newest version of **Idrisi** is available from http://www.clarklabs.org/.

The same map sheet must have the same boundary coordinates in all data layers. Alternatively, the minimum X coordinate, minimum Y coordinate and the length of the edge of a map sheet are enough if raster files of all map sheets have the same number of rows, columns, and pixels.

The specific terms used in this tutorial are listed in Appendix 1. SQL commands from this tutorial, a detailed description of **Constud**, list of publications and source code of some functions vital to understanding the algorithm are available from the **Constud** website http://www.geo.ut.ee/CONSTUD. The compiled application is freeware and available for installation and launching from the website. This website also reflects the latest changes in the software system and errata of this tutorial. Once installed, the software checks for updates before every launch. The assembly name may contain a version number, e.g. **Constud 2**.

1.1. Exercises and example data

Tasks comprising examples of how to use **Constud** ***are in bold letters and numbered***. The numbers cross chapters because it is not possible to complete most tasks prior to completing previous tasks. In general, use of **Constud** should begin with the planning of experiments, selection of variables and preparation of data layers. Basic knowledge of geoinformatics, statistical data processing and **Access** or **SQL Server** database management systems is required.

The types of tasks are illustrated by the following icons:

 — launch **Constud** program or start calculations.

 — select settings in **Constud**.

 — press button to open the next window.

 — create or copy a table or a view.

 — alter a database object (except when selecting the dependent variable).

 — switch on/off dependent variables in the field *[DEP_VAR].[calc]*.

 — add data to a database table or files to the computer.

 — check settings and data.

 — learn and give meaning to the results; try it yourself.

The tasks related to the **Constud** database are predominantly written as SQL commands in this tutorial. Both **SQL Server** and **Access** database management systems offer interactive tools for the same operations using a mouse and keyboard in a graphical environment, such as design views, table views and graphical wizards. You can use these tools instead of SQL, if you prefer. SQL is preferred in this text only because it depends less on the database management system and its version, and is easier to integrate with other instructions given in text format. Capital letters can be used freely in names of database objects and SQL terms — SQL is not case sensitive. *The names of database objects* in SQL commands and in the text, if consisting of a table name and a field name are in square brackets — this enables to use reserved words and extra symbols in object names.

Terms defined in Appendix 1 are highlighted. The definition is displayed by clicking the term; *Alt+Left* keys return reading to the main text.

Tutorial data needed to solve the exercises are available for download from the **Constud** website. Tabular data are found in the **Access 2003** database *Constud_*

tutorial_data.mdb. Raster data layers are located in the archive of digital spatial data at the Institute of Ecology and Earth Sciences (ADSD). Information about ADSD is available at http://digiarhiiv.ut.ee. The files in ADSD are accessible to download only by registered users. The data layers can be built up and located elsewhere but must be organised in a predefined order (*Remm et al., 2010*).

The FTP server address and the authentication string for ADSD are included in the compiled version of **Constud** as default values. Users with an account at the University of Tartu who are located outside the university's local network should use a VPN connection (http://arvutiabi.ut.ee/pages/viewpage.action?pageId=3309798). Users from other institutions should mount their own data layers or obtain access to a data source. Instructions for preparing data layers are found in the technical description (*Remm et al., 2010*).

The following exercises for using **Constud** are explained in this tutorial:

1. Creating a **Constud** database.
2. Calculation of pattern indices to database tables.
3. Calculation of pattern indices to raster file (square shape maps or rectangular maps).
4. Fitting of weights for features and for cases (spatial and non-spatial data).
5. Fitting the weights for recognition of a single category of a nominal dependent variable.
6. Calculation of validation fit for the best predictive set.
7. Re-selection of exemplars.
8. Prediction of the dependent variable to a database table.
9. Estimation of the similarity to a given category.
10. Prediction to a binary raster.
11. Log table browsing and log data interpretation.

Each chapter ends with questions. The answers are found in Appendix 4.

1.2. Database Structure

Constud is designed mainly for estimating the values of a dependent variable according to the values of explanatory variables (features) in a learning dataset. The **Constud** database includes a table for attributes of dependent variables named *DEP_VAR*, and a table for attributes of features — *EXPL_VAR*. Table *DEP_VAR* is the primary object in the **Constud** database — **Constud** first reads data from this table

when starting a new task or learning session. **Constud** checks which dependent variable it has currently to deal with from *DEP_VAR* (field *calc*), reads the identifier (*FID*) of the variable, the name of the database object (table or view) where the learning data and selected exemplars are found, and some other attributes depending on the selected operation.

A simple database view is a stored select query that is accessible as a virtual table.

Each record in the *DEP_VAR* table must have a matching field in the *EXPL_VAR* table — a field in which the name starts with *F* followed by the FID number (Fig 1-1). **Constud** turns to the *EXPL_VAR* table and reads field *F* followed by the FID number; e.g. if FID = 32 then *[EXPL_VAR].[F32]*. **Constud** will use features which have the value *True* in this field. Each feature is defined by a set of attributes. The obligatory attributes are the feature identifier (*AID*), the data layer identifier (*KID*) from which the feature was calculated and spatial index (*index_ID*) that is related to the table *SP_indices* (Fig 1-2).

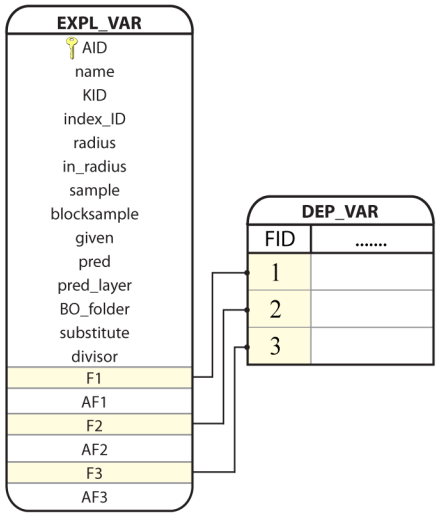


Figure 1-1. Attributes of features in the *EXPL_VAR* table and relationships between tables *EXPL_VAR* and *DEP_VAR*. Fields of *DEP_VAR* are listed in Appendix 2.1.

A data layer must be connected to each feature because when using a spatial variable, the temporal limits in which the feature can be used are defined for data layers but not for every single feature. The temporal interval of a feature is compared to the temporal interval of an observation. The feature is used for calculating similarity between observations only if the validation interval of the feature overlaps the validation interval of both observations (Fig 1-3). In case of non-spatial data, the difference between nominal and numerical features is made according to the data

layer related to the feature. Other attributes of data layers are needed only for calculating spatial indices.

A data layer must be connected to each feature, even in case of non-spatial data.

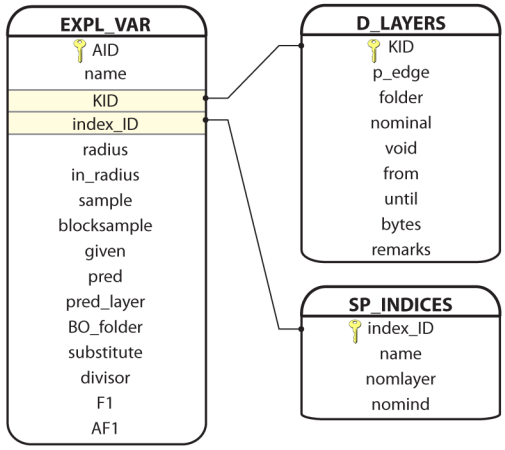


Figure 1-2. The attributes of features in the *EXPL_VAR* table and relationships between tables *EXPL_VAR*, *D_LAYERS* and *SP_indices*.

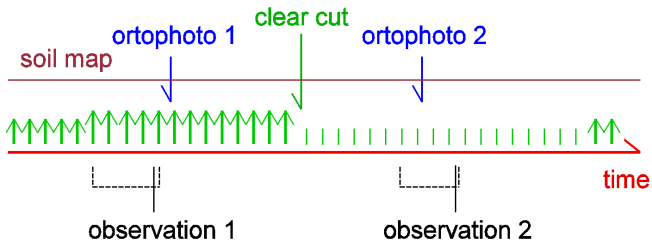


Figure 1-3. Two observations and their validation intervals (dashed line). The validity of soil map is unlimited but both ortophotos are valid only for the date of the shot. Observation 1 can be combined with features calculated from the soil map; observation 2 can be combined with features calculated from the soil map and from ortophoto 2.

Index_ID is the second obligatory attribute of spatial features since the difference between nominal and numerical spatial features is made according to the index. The difference between nominal and numerical non-spatial features is made according to the data layer attributed to the feature. The rule for calculating partial similarity depends on the type of the feature. Calculating difference between codes of categories of a nominal variable has no meaning. Even for features that already have values, the index ID is obligatory because the type of feature is derived from the index.

When calculating spatial indices, a radius is the third obligatory attribute of a feature.

> The three basic attributes of a spatial feature in Constud are: data layer, index and radius.

Constud reads the folder name, type (nominal or numerical), number of bytes per pixel, length of pixel edge, value for void and the temporal limits of the data layer from the *D_LAYERS* table. A prefix defined in **Constud**'s main window is added to the folder name (Fig 1-4). If, for instance, all data layers are located in subfolders of *C:/Constud/Layers/,* then it is enough to insert the folder name *soiltypes* into the table *D_LAYERS* for a data layer located in *C:/Constud/Layers/soiltypes*.

Figure 1-4. The text box for folder prefix input.

Results of learning iterations are recorded into log tables. Log tables are obligatory for every dependent variable applied for a learning or prediction task. The attributes of learning iterations recorded in log tables are as follows:

1) the given category for learning categories of a nominal variable separately (*given*),

2) the date and time of completing the iteration (*date*),

3) the learning fit obtained in a learning sample (*t_fit*),

4) the validation fit obtained by applying the predictive set of exemplars and features for the validation sample,

5) the sum of similarity sought for decision (*sumsimmax*) controlling the number of exemplars used for decision,

6) the number of exemplars (*etn*),

7) sample size (*sample*),

8) the weights for features (*F_weights*).

ID number of the hitherto best iteration according to validation fit is recorded to field *[DEP_VAR].[log_ID]*. The field *ID* connects tables *DEP_VAR* and *LOG_F…* (Fig 1-5).

The user can alter values and add data to tables while **Constud** operates within the same database. Changing attributes in the database is a way to control machine learning in **Constud** without interrupting it.

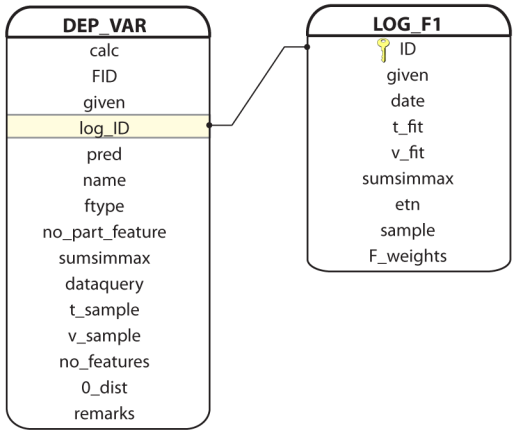


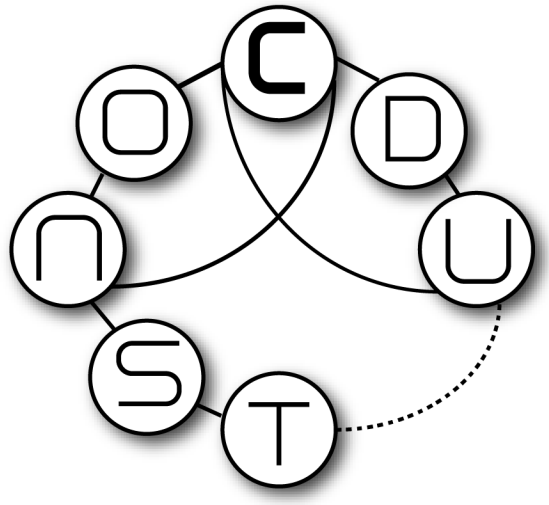
Figure 1-5. Attributes of dependent variables in the *DEP_VAR* table, in log table of the variable F1 and the relationship between these tables.

Questions

1. Can **Constud** calculate indices of spatial patterns from vector format polygons?

2. Can **Constud** produce a map depicting distance to the nearest running water body?

Data structure 2



2.1. Database

Constud operates only in conjunction with a database that has a specific structure. The required database objects and their composition are listed in Appendix 2. The database can be in one of the following formats:

1) **Access 2000, 2002–2003, 2007–2010,**

3) **MS SQL Server,**

4) **MS SQL Server Express** (free download: http://www.microsoft.com/express/Database/),

5) **MS SQL Server Compact Edition.**

SQL Server CE 3.5 SP2 is freely available from: http://www.microsoft.com/sqlserver/2005/en/us/compact-downloads.aspx. A newer version, 4.0, is not yet fully integrated with **SQL Server Management Studio 2008 R2** and **.NET Framework 4**; therefore, the current version of **Constud** does not support **SQL Server CE 4.0**. **SQL Server CE** databases reside in a single sdf-file, which can be up to 4 GB in size. The sdf-files can be produced and managed with **Microsoft Visual Studio** and **SQL Server Management Studio**. A list of **SQL Server CE** third party tools is available at http://erikej.blogspot.com/2009/04/sql-compact-3rd-party-tools.html. **SQL Server CE** does not support saved queries and database views.

Do not protect **Access** and **SQL Server CE** files with a password while working with **Constud**.

Constud is not able to communicate with password-protected Access or SQL Server Compact databases.

An SQL Server is a good fit if several parallel applications are planned, when the database contains gigabytes of data, or when database usage over a network is planned. If speed is the priority then **SQL Server CE** might be the best choice since **SQL Server CE** runs from memory. It is easier to use **MS Access** when you have just started learning **Constud**. The following instructions for practical exercises have been adjusted primarily for **Access** users.

1. Create a new empty database or decide which existing database to use.

The possible database options are listed above. Decide which one you would like to use. Perhaps it is easier to use a local **Access** database than a server database, although the last option might be more interesting for a dedicated user to learn.

Connection to the **SQL Server Express** database *CS1* in computer Vinson is the

default option in **Constud**'s current version. You can use it while working within the local network at the Department of Geography, University of Tartu. Use **SQL Server Management Studio** (free download: http://www.microsoft.com/express/Database/InstallOptions.aspx — *Management Tools* option*)* to connect to this database or to work with any other **SQL Server** database.

Ask the database administrator to obtain a user account for an existing server-based database.

2.2. Data tables

When the **Constud** database exists and you have administrative rights to access it, you can start preparing database objects (tables, views and queries). The required tables can be created using **Constud** (Chapter 2.2.1) or imported from an existing **Constud** database. A few of the previously tested databases are available from the **Constud** website.

2.2.1. Creating data tables using Constud

If the previous version of the **Constud** software is installed on your computer then uninstall it. After that, either install or launch **Constud 2** using links at http://digiarhiiv.ut.ee/Constud2/DW_eng.aspx , or start the previously installed instance of **Constud 2**.

2. Launch Constud. Select the English interface (Fig 2-1).

After the language has been selected, the **Constud** main window opens. The window has four tab panels: *Spatial Indices*, *Learning*, *Knowledge Test* and *Knowledge Application*. The attributes of database connection can be set in the lower part of the main window (Fig 2-2).

3. Select the Constud database and an appropriate connection.

4. Click the button with text 'Create tables'.

The window for creating data tables opens (Fig 2-3). **Constud**'s main tables can be created one by one or all at once.

If the message: *Cannot read knowledge base* appears then check whether the database name and connection provider are correct. Closing and reopening the created database or database connection may also help. Another message reading: *Cannot create table…* is predominantly caused by an already existing database object of the same name. If the table creation was completed but nothing changed

in the database, then compress and repair the database.

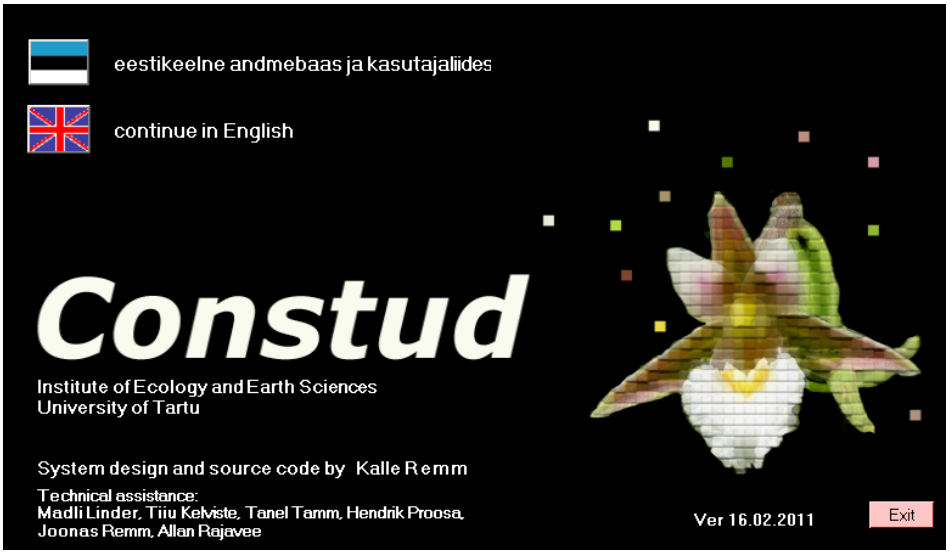


Figure 2-1. **Constud** greeting window.

Figure 2-2. Attributes of data connection (all options visible) in the lower part of the **Constud** main window.

If using **SQL Server** or **SQL Express**, decide either to use an existing or create a new database schema. The default schema in these systems is *dbo*. Database schema is a section of a database — a set of user rights and database objects. Identical table names can occur simultaneously in different schemas within the same database.

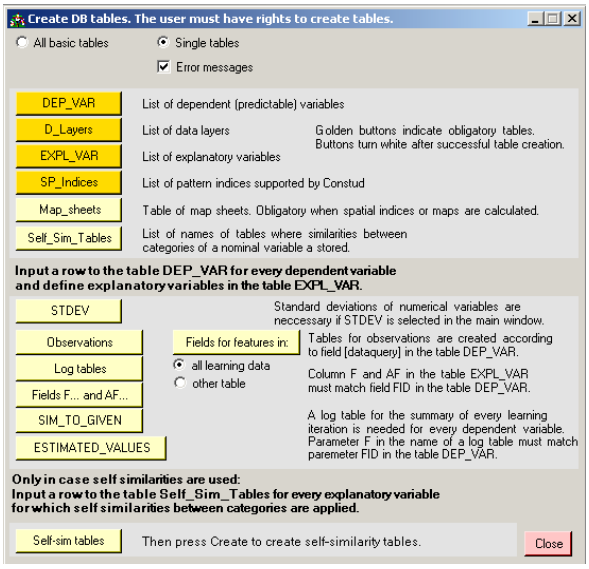
5. Create the basic tables in the Constud database one by one (Fig 2-3 B) or simultaneously (Fig 2-3 A,C).

Visibility of the button *All basic tables* depends on the radio button selected in the upper part of the form for creating tables. The basic tables are always necessary for **Constud** to operate. Additional tables are needed for some operations and for some types of data.

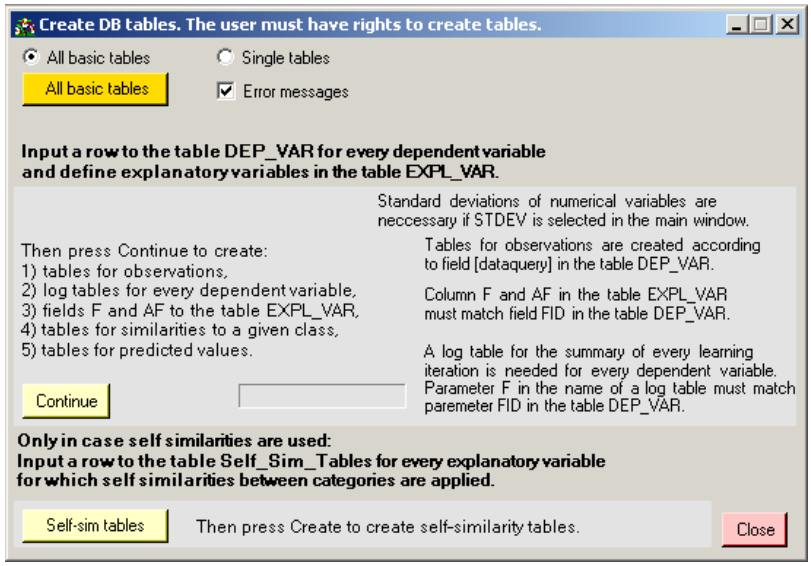
Leave **Constud** open. If using an **Access** database — press the button: *Compact and repair database*; in **SQL Server Management Studio** — press *Refresh* from the drop

down menu that opens after the right mouse button click on the section *Tables*.

A)



B)



C)

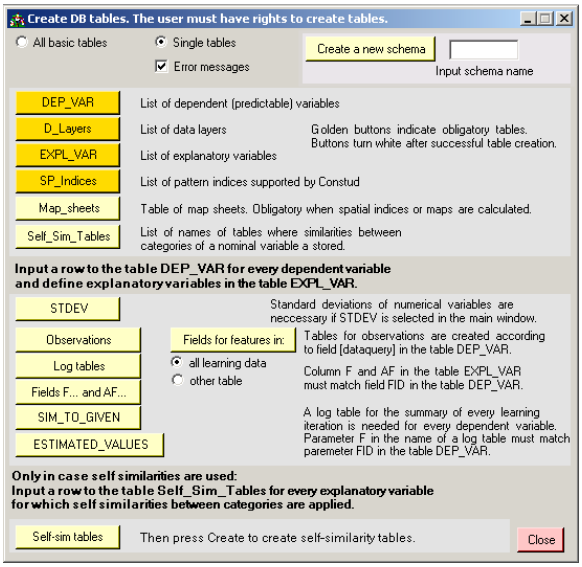


Figure 2-3. Constud window for creating database tables. **A** — Access: main tables separately, **B** — Access: main tables in one click, **C** — SQL Server: main tables separately.

6. Insert the characteristics for a dependent variable in the newly created DEP_VAR table.

The first **Constud** exercise is a calculation of spatial pattern indices. Copy the next SQL command to the *SQL window* if using **Access**.

```
INSERT INTO [DEP_VAR]
([calc],[FID],[given],[log_ID],[precl],[name],[ftype],
[no_part_feature], [sumsimmax], [dataquery], [t_sample], [v_sample],
[no_features], [0_dist], [remarks]) VALUES
(False,1,0,0,0,'calculation of spatial indices',0,0,1,
'OBSERVATIONS1',100,100,10,0,NULL);
```

If using an **SQL Server**, **SQL Server Compact Edition** or **SQL Express**, add an apostrophe before and after the words *True* and *False*; for **SQL Server** or **SQL Express** add the database and schema name (if these are not the default values) to the name of each table and view mentioned in SQL commands and queries.

In SQL Server add an apostrophe before and after the words True and False, and the schema name before the table name.

```
INSERT INTO [CS1].[dbo].[DEP_VAR]
([calc],[FID],[given],[log_ID],[precl],[name],[ftype],
 [no_part_feature],[sumsimmax],[dataquery],[t_sample],
[v_sample],[no_features],[0_dist],[remarks]) VALUES
('False',1,0,0,0,'calculation of spatial indices',0,0,1,
'OBSERVATIONS1', 100,100,10,0,NULL);
```

The meaning and format of the database fields mentioned in this tutorial is given in Appendix 2 and also in the technical description of the software system (*Remm et al., 2010*). The type of a dependent variable is coded in the *DEP_VAR table* as: 0 — multinomial, 1 — reserved for multidimensional numerical, 2 — numerical, 3 — binomial, 4 — single category of a multinomial variable opposed to all other categories of the same variable. As of now, the multidimensional numerical variable is not ready for implementation in **Constud**.

The type of a dependent variable is coded as:
0 - multinomial, 2 - numerical, 3 - binomial, 4 - single category of a multinomial variable opposed to all other categories of the same variable.

7. Open the EXPL_VAR table and insert the characteristics of the first descriptive feature.

```
INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl)
VALUES (1,'share of forest', 1,1,50,0,1,False,64,False);
```

The given value in this query, 64, marks woodland in the Estonian Basic map. According to this command, the first feature has the identifier AID = 1, its name is '*share of forest*', this variable is calculated from the data layer KID = 1, using the index number 1 (*index_ID* = 1, which is the share of a given category). The share is calculated within a radius of 50 units (*radius* = 50) and the internal radius is zero (*in_radius* = 0), which means that the annulus kernel is not used, the code of given category is 64, and limiting polygons and the block kernel are not used.

8. Add two more features.

Change the previous SQL command using the following values:

AID = 2, KID = 2, index_ID = 102, radius = 300, in_radius = 200 and

AID = 3, KID = 3, index_ID = 115, radius = 50, in_radius = 0.

Give understandable names to these variables.

The following SQL commands serve as examples.

```
INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl)
VALUES (2,'altitude in neighbourhood', 2,102,300,200,1,False,Null,
False);

INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl)
VALUES (3,'share of forest', 3,115,50,0,1,False,Null,False);
```

9. Insert the three data layers (KID = 1...3) mentioned in the previous queries in the D_LAYERS table.

In ADSD (http://digiarhiiv.ut.ee) these data layers are areal categories of the Estonian Basic Map (*PK-uusim*), with surface elevation given in metres according to the Shuttle Radar Topography Mission global elevation model (*SRTM_elev_byte*), and land elevation in decimetres digitised from topographical maps (*elev_dm*). Notice that the third layer has two bytes per pixel; the others have three. The insert queries should look like the following. The meaning and format of fields are listed in Appendix 2.5.

```
INSERT INTO [D_LAYERS] ([KID],[p_edge],[folder], [nominal],[void],
[from],[until], [bytes],[remarks])
VALUES (1,5,'PK-uusim',True,0,null,null,1,null);

INSERT INTO [D_LAYERS] ([KID],[p_edge],[folder], [nominal],[void],
[from],[until], [bytes],[remarks])
VALUES (2,100,'SRTM_elev_byte', False,0,null,null,1,null);

INSERT INTO [D_LAYERS] ([KID],[p_edge],[folder], [nominal],[void],
[from],[until], [bytes],[remarks])
VALUES (3,10,'elev_dm',False,0,null,null,2,null);
```

Close all database objects and continue creating tables using **Constud**.

10. Press the Continue button if creating tables simultaneously or press buttons: Observations, Log tables, Fields F and AF, SIM_TO_GIVEN and ESTIMATED_VALUES if you decided to create tables separately.

Tables to store estimated similarity to a given category are created only for dependent variables belonging to type 4. Instructions regarding how to create these tables are included with the exercise for calculating similarity to a given category (Chapter 4.5). The buttons turn pale when the target table is ready. You can see the new table in the database that was connected to **Constud**. The database may need to be refreshed to show new objects.

In general, when similarity is estimated between observations according to nominal

variables, category matching is counted as similarity and different categories as dissimilarity. Quite often, more flexible counting of similarity is justified. **Constud** enables one to differentiate between similarities for categories of the same nominal variable (self-similarity), and to store self-similarity values in database tables. A separate table must be created for every nominal variable for which self-similarity is applied. Names of self-similarity tables must be recorded in the *Self_sim_tables*. Similarity between categories is assumed to be zero if it is not defined. An example of a self-similarity table is found in the database of tutorial data *Constud_tutorial_data.mdb* in which the *OmasarnasusPK table* contains similarities (%) subjectively assigned to a few combinations of the Estonian basic map areal categories. According to this table, similarity between two observation sites is assigned at 10% if one belongs to a category *forest* and the other to a *young forest*.

The database structure necessary for **Constud** is ready now. The next task is to fill the structure with data.

11. Add example data and coordinates of observation sites to the table of observations and also to the table of predictable cases.

A set of observation sites can be found in the *Constud_tutorial_data.mdb database.* Let's use the presence and absent sites of an orchid *Epipactis palustris* from the table *E_palustris*. The data import function can be found as follows.

If using **SQL Server software**, click the right mouse button on the database name and select *Tasks* => *Import Data*.

In **Access 2003** drag the object or choose *Menu* => *File* => *Get External Data* => *Import*.

In **Access 2007** drag the object or choose *Menu* => *External Data*. Then select **Access 2003** files and find the file entitled *Constud_tutorial_data.mdb*.

Import table *E_palustris* to a new table with the same name. When the table has been imported, use an insert query to copy data from the imported table into the table of observations used by **Constud**. The following SQL command is an example. The field *[E_palustris].[occurs]* contains presence/absence records of the species at observation sites. Presence/absence records should be copied into the field *[OBSERVATIONS1].[F]*, and *X* and *Y* coordinates to the same fields in *OBSERVATIONS1*.

```
INSERT INTO[OBSERVATIONS1] (VID,[F],[X],[Y])
SELECT [VID],[occurs],[X],[Y] FROM [E_palustris];
```

Refresh the database and check whether the data reached fields: *VID*, *F*, *X*, *Y* in the table of observations. If so, remove the imported *E_palustris* table but leave the **Constud** table of observations.

12. Add the numbers and border coordinates of map sheets to the Map_sheets table.

The numbers and border coordinates of the Estonian 1: 20,000 basic map sheets are available in the *PK* table in *Constud_tutorial_data.mdb*. Use different sheet numbers and coordinates if using another nomenclature. The units and the measurement system of coordinates must match between observations and map sheet boundaries.

You can use an SQL command similar to the previous task to transfer map sheets data — only the table and field names are different. You have to add database and schema names before the table names in **SQL Server** (if these are not the default values).

```
INSERT INTO [Map_sheets] ([NR],[min-x],[min-y],[max-x],[max-y])
SELECT [NR],[min-x],[min-y],[max-x],[max-y]
FROM [PK];
```

13. Check whether the map sheet data reached the Map_sheets table and then delete the imported PK table.

Use the *Validate tables* button in **Constud**'s main window to check the data structure (Fig 2-2). The database created is needed for the next tasks in this tutorial.

To conclude the issue of creating data structure using **Constud** tools: think twice — it may be easier to make a copy of an existing **Constud** database, replace the data in it, and adapt it for a new **Constud** task.

Tables in a Constud database can be created using Constud tools, by copying single objects, or by copying and modifying the whole database.

Questions

1. Can **Constud** interoperate with an **Oracle** data management system?

2. Can **Constud** interoperate with a **MySQL** data management system?

3. How many indices are listed in the *Sp_indices* table?

4. What is the difference in the *Index_ID* numbers of indices calculated from nominal data layers and those calculated from numerical data layers?

5. Which indices of a spatial pattern in **Constud**'s *Sp_indices* table have nominal values? In what way are these common?

6. In which field is the name of the database object recorded wherein the values of explanatory variables are stored?

7. What is the maximum number of tables for similarities between categories of the same nominal variable used in **Constud**?

8. In which fields must measurement units of distances and spatial coordinates be the same in a **Constud** database?

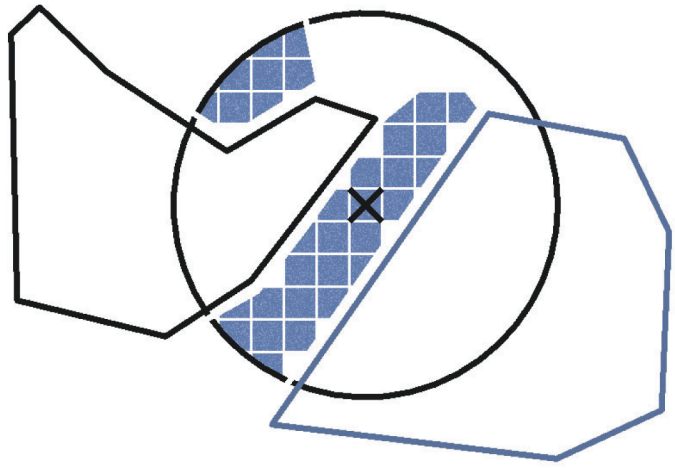
9. Where is the ID number of the best result of machine learning recorded in addition to a log table?

10. Which data are recorded in the field *[D_LAYERS].[p_edge]*?

11. Can **Constud** use data in which the location of observation sites is recorded in kilometres?

12. Can **Constud** use data wherein the location of observation sites is recorded in degrees of geographical coordinates?

Pattern Indices 3



The definitions of pattern indices as features of a location are given in the technical description (*Remm et al., 2010*), and in Appendix 2.9 and 3. Values of all indices are stored in byte format in the **Constud** system, even if calculated from an integer format data layer. Algorithms of some indices (e.g. Shannon's entropy) are modified to yield byte format values. Examples of the application of spatial indices are on the web page of **LSTATS** software (http://www.geo.ut.ee/LSTATS). The same algorithms of indices are implemented both in **Constud** and in **LSTATS**,

The maximum value in byte format, 255, universally stands for undetermined feature values in **Constud**. For example, slope direction cannot be calculated for a horizontal surface.

The values of explanatory variables are integers between 0 and 254; a value of 255 denotes missing data.

14. Change the value in field [DEP_VAR].[calc] to True.

The display format of the value *True* differs according to user interface settings. The common alternatives are: *-1*, *1*, *Yes*, *On* or a *Checked box*.

All tasks in **Constud** start from the *DEP_VAR* table. The *True* values in *[DEP_VAR].[calc]* direct **Constud** to deal with the dependent variables defined by that record. There is only one dependent variable defined in the *[DEP_VAR]* table until now, if the exercises are followed according to this tutorial. We will add other dependent variables in later tasks. For calculating indices, only the fields: *FID*, *calc* and *dataquery* are relevant in the *DEP_VAR* table. The *FID* field is related to the *EXPL_VAR* table (Fig 1-1); the *calc* field determines which variables to calculate, and *dataquery* indicates where the coordinates and feature values are stored.

15. Open the EXPL_VAR table and mark field F1 as True in all three existing rows.

The field is *F1* because the FID-number of the selected dependent variable is 1. Now, the dependent variable number 1 is connected to features defined earlier. If you do not find the field *[EXPL_VAR].[F1]* in your **Constud** database then it means step 10 has not been completed successfully. Use the *Fields F... and AF...* buttons in the **Constud** form *Create tables*.

3.1. Indices to a database table

Most errors made while using **Constud** are caused by inconsistencies in the **Constud**

database. So before calculating indices, check every time that:

1) the correct row and only the correct row is switched on in the *DEP_VAR* table;

2) the calculated features and only these features are selected in the *F…* field (the ellipses correspond to the FID number of the selected dependent variable) in the *EXPL_VAR* table;

3) the fields for computed features exist in the output table (by default the table whose name is in the field *[DEP_VAR].[dataquery]*);

4) the metadata and folder names of data layers are correct.

In addition, use the *Validate tables* button in **Constud**'s main window to check the data structure (Fig 2-2).

16. Launch Constud. Select tab panel Spatial Indices and the local or FTP path to the data layers from which the indices will be calculated (Fig 3-1). Select the database you prepared in the previous chapter and press the Continue button.

Students at the University of Tartu can use the default path to data layers either via local area network (LAN) or using VPN connection. Other users have to arrange the data layers themselves. See: technical description of **Constud** (*Remm et al., 2010*).

Figure 3-1. Settings in the **Constud** main window for calculating spatial indices from raster files on an FTP server to a table in the **Constud** database. Cells *login* and *psw* contain hidden credentials for accessing data in ADSD. Insert proper credentials for other FTP servers.

Ensure that fields *[1]*, *[2]* and *[3]* in *EXPL_VAR* corresponding to explanatory variables 1, 2 and 3 are filled with the values of these features at all observations. Recall that feature 1 was defined as the proportion [%] of forest within a radius of 50 m, feature 2 is the average surface elevation according to SRTM data at a distance of 200–300 m, feature 3 is the relative elevation of the location compared to the mean elevation plus 100 within 50 m (observation sites on negative land forms have values of the third feature less than 100, positive land forms > 100). Units of the values in the last two fields are the same as in source data — metres for the SRTM data and decimetres for the data layer *elev_dm*.

3.1.1. Calculation of indices using a sample

Calculation of complicated spatial indices for a large number of sites and within a larger radius containing a lot of pixels can be rather time consuming. The problem is mainly related to indices, for which the algorithm includes a comparison of values in pairs of pixels, because the number of all possible pairs is = $N(N-1)$ if the number of objects is N. Fortunately, a modest sample containing some hundred pixels is usually representative enough. Wasting resources to calculate indices of a spatial pattern using thousands of pixels at every location is not reasonable.

The probability of including pixels in a sample is defined in the field *[EXPL_VAR].[sample]* as a single format real number. Random sampling can be combined with annulus kernel and block sampling (Fig 3-2). The use of block sampling is set in the field *[EXPL_VAR].[blocksample]*. Block sample means the selection of 3 × 3 pixel blocks and enables to use more neighbouring pixels than a simple random sample that has the same sample proportion. The distance between blocks depends on the proportion of the sample — the number of selected pixels matches the predetermined share as precisely as possible. A block sample is useful for calculating indices in which the comparison of adjacent pixels is involved and when the sample proportion has to be relatively small due to a large number of pixels within a kernel. Neighbouring pixels are compared, for example, when calculating gradient smoothness and Moran's *I* of 8 neighbouring pixels.

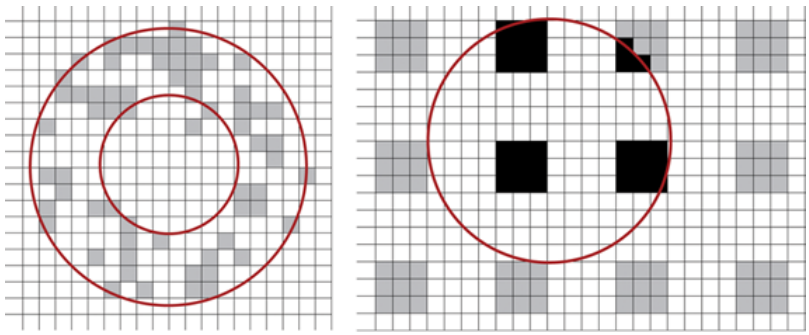


Figure 3-2. Random sample of pixels in an annulus kernel and a block sample in a round kernel.

To calculate a new feature, you have to add the following fields and values to the **Constud** database.

17. Insert two additional feature definitions to the table EXPL_VAR; check the field F... of these records where the ellipses correspond to a checked (calc = True) record in DEP_VAR.

Notice that the new features differ from the existing feature number 1 only by the sample share. The feature number 1 was calculated within 50 m from the rasterised

basic map where pixel edge is 5 m. The number of 5 m pixels in a 50 m circular kernel is 317, except in cases closer than 50 m to the border of a map sheet. The next queries insert two additional features; both are modifications of the first feature: one uses a sample share of 50%, the other 10%.

```
INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl, precl_layer, BO_folder, substitute, divisor,F1)
VALUES (4,'share of forest 50%' ,1,1,50,0,0.5,False, 64, False,
null,null,null,null,True);

INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl, precl_layer, BO_folder, substitute, divisor,F1)
VALUES (5,'share of forest 10%' ,1,1,50,0,0.1,False, 64, False,null,
null,null,null,True);
```

Mind apostrophes before and after the words *True* and *False*, and a schema name before the table named *EXPL_VAR* if using an **SQL Server**.

18. Switch off calculation of the previous features.

You can use the following SQL command if using an **Access** database.

```
UPDATE [EXPL_VAR] SET [F1]= False WHERE [AID]<4;
```

Mind apostrophes and schema name if using an **SQL Server**

19. Add fields [4] and [5] to the table of observations (the table or view, whose name is in the field *[DEP_VAR].[dataquery]*). Use the button *Fields for features* in the *Create tables* window or use the following SQL command.

For **Access** database:

```
ALTER TABLE [OBSERVATIONS1] ADD [4] byte NULL, [5] byte NULL;
```

Adjust the database and schema names if using an **SQL Server**. Note also that the *byte* format is called *tinyint* in **SQL Server**.

Check whether fields *[4]* and *[5]* really exist.

20. Launch Constud to calculate values for features number 4 and 5 by pressing Continue in Constud's main window.

Compare the values for feature number 1 with the values for feature numbers 4 and 5, which differ only by using respectively 50% and 10% samples of pixels in the kernel.

3.1.2. Calculation of indices using limiting polygons

Sometimes indices of a spatial pattern have to be limited to given polygons. For example, the mean reflectance of forest canopy at an observation site near a forest's edge should be calculated using pixels from a remote-sensing data layer only up to the forest boundary. Pixels within a given radius that do not represent forest, but a field that is nearby, should be omitted from the calculation. Limiting polygons are also useful when the characteristics of the described sites are needed and these features should be calculated only within the valid boundaries of the description. To conclude, pixels located within a given radius and belonging to the same category in the layer of limiting polygons, as well as the focal pixel, are included in the sample when a layer of limiting polygons is applied (Fig 3-3).

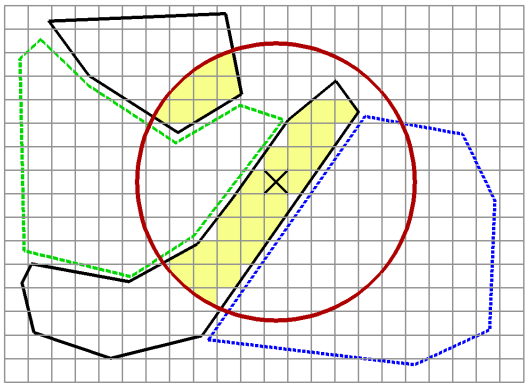


Figure 3-3. The effect of limiting polygons to the selection of pixels (grey squares). Each polygon represents a category of the spatial pre-classifier; observation site (x) is within the category with a black boundary; the circle indicates the limiting radius defined for this explanatory variable. Only the pixels within the focal category and within the radius are included in the sample (pale yellow fill). Pixels are included according to the location of their centre.

21. ***Add features 6 and 7*** for the standard deviation of pixel values in the Landsat 7 ETM+ image from 23.05.2005. For feature number 7, the kernel is delimited by the same areal category as at the location of the observation. The data layer of this Landsat image is in folder *ETM-186_019-23-05-2005-B8* in ADSD. Pixel edge in the data layer is 10 m, the radius of pixel inclusion should be 50 m for both features, and data layer number 1 should be defined as the layer of limiting polygons for feature number 7. The code for standard deviation is 103, as you can see from the table *SP_indices*.

The additions to the **Constud** database are similar to operations 9, 17–19. Look carefully at the following SQL examples. The examples are valid for an **Access** database; for an **SQL Server** the syntax has to be modified a little (add schema name, apostrophes before and after *True* and *False*, and *tinyint* for *byte* data format).

Register a new data layer for the Landsat greyscale image:

```
INSERT INTO [D_LAYERS]
([KID],[p_edge],[folder],[nominal],[void],[from],[until],[bytes],
[remarks]) VALUES (4, 10,'ETM-186_019-23-05-2005-B8',False,0, null,
null, 1, null);
```

Add feature definitions:

```
INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl, precl_layer, BO_folder, substitute, divisor,F1)
VALUES (6,'SD of ETM+ image',4,103,50,0,1,False,0,False,null,null,
null,null,True);

INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl, precl_layer, BO_folder, substitute, divisor,F1)
VALUES (7,'SD of ETM+ image',4,103,50,0,1,False,0,True,1,null,null,
null,True);
```

Add fields for calculated values:

```
ALTER TABLE [OBSERVATIONS1] ADD [6] byte NULL, [7] byte NULL;
```

Check whether fields *[6]* and *[7]* really exist.

Switch off calculation the previous features:

```
UPDATE [EXPL_VAR] SET [F1] = False WHERE [AID] <6;
```

22. Launch Constud to calculate values for features 6 and 7.

Compare the calculated values of these features.

3.1.3. Calculation of indices from seamless data layers

If the checkbox *seamless* (Fig 3-4) is checked and if the outer radius for pixel inclusion around the observation site exceeds the limits of the map sheet where the observation site is located, then **Constud** includes pixels from neighbouring map sheets.

23. Add two features: 8 and 9 with identical attributes, except AID. Let the index be calculated from data layer 1 and the feature be the number of categories (index_ID = 6) within 100.

The SQL commands are as follows.

Add feature definitions:

```
INSERT INTO [EXPL_VAR] (AID,name,KID,index_ID,radius, in_radius,
```

```
sample, blocksample, given, precl, precl_layer, BO_folder,
substitute, divisor,F1) VALUES (8,'number of categories',1,6,100,0,
1,False,0,False,null, null,null,null,True);

INSERT INTO [EXPL_VAR]
(AID, name, KID, index_ID, radius, in_radius, sample, blocksample,
given, precl, precl_layer, BO_folder, substitute, divisor,F1)
VALUES (9,'number of categories seamless',1,6,100,0,1,False,0,False,
null,null,null,null,False);
```

Add fields for calculated values to the table of learning observations.

```
ALTER TABLE [OBSERVATIONS1] ADD [8] byte NULL, [9] byte NULL;
```

Check whether fields *[8]* and *[9]* really exist.

Switch off calculation of the previous features:

```
UPDATE [EXPL_VAR] SET [F1] =False WHERE [AID] <8;
```

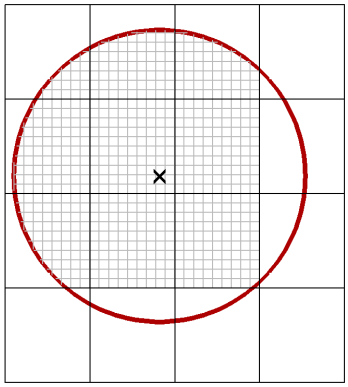


Figure 3-4. Including pixels from the first order neighbouring map sheets if the option *Seamless data layers* is selected. Cross — observation site, circle — kernel radius, black squares — map sheets, grey grid — included pixels. Pixels from the second order neighbouring sheets are excluded.

24. Launch Constud to calculate feature 8 without selecting the seamless option. After calculating the feature values, turn the value [EXPL_VAR].[F1] = False where [EXPL_VAR].[AID] = 8, and = True where AID = 9.

Close *EXPL_VAR* to be sure the changes are saved.

The next step is a little time consuming — you can leave **Constud** working and do other things in the meantime.

25. Start Constud to calculate feature 9 using the seamless option from the Constud main window (Fig 3-5).

Notice that calculation from a seamless data layer is much slower. Compare the calculated values of features 8 and 9.

Switch off the seamless data layers option or close **Constud**.

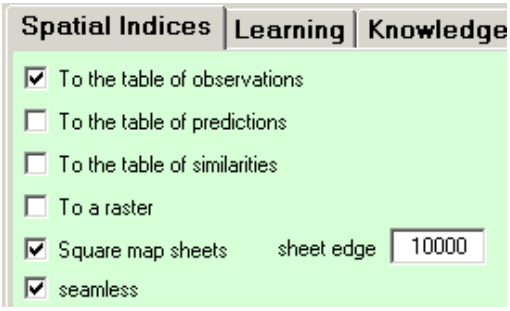


Figure 3-5. Settings in the **Constud** main window for calculating features from seamless data layers.

3.1.4. Calculation of indices to the predictions table

Until now, the output table or view in the **Constud** database was defined in *[DEP_VAR].[dataquery]*, which is the database object containing records of learning observations. For a prediction and validation task, a separate dataset of predictable values as a separate table or view must be prepared. For deducing the predicted values, these predictable observations must also have values of the same features as in the learning data.

26. Import or copy the VID numbers and coordinates of predictable sites from the Predictable_sites table located in the database of tutorial data (Constud_tutorial_data.mdb) into the previously created predictions table in the Constud database. The predictions table should be named ESTIMATED_VALUES_1.

Steps 11 and 12 were similar and can serve as examples for import and insert tasks.

27. Add fields for values of the new features to the predictions table.

The table should include the three first variables following this tutorial. The other features can be added manually, using the next query or by using the button *Fields for features in:* combined with the option: *other table* in the **Constud**'s *Create tables* window.

```
ALTER TABLE [ESTIMATED_VALUES_1] ADD
[4] byte NULL,
[5] byte NULL,
[6] byte NULL,
[7] byte NULL,
[8] byte NULL,
[9] byte NULL;
```

Check whether fields 1…9 are present in the table.

28. Switch on all features in [EXPL_VAR].[F1].

```
UPDATE [EXPL_VAR] SET [F1] = True;
```

29. Launch Constud, check the option box: To the table of predictions (Fig 3-6) and insert the name of the database object where you intend to store the calculated values (following this tutorial: ESTIMATED_VALUES_1). Start calculation by pressing Continue.

Check the calculated values.

Figure 3-6. Constud settings for calculating spatial indices to the predicted values table.

There is also an option in the **Constud** main window for calculating spatial indices to the table wherein the similarity to a given category is stored. It only offers an option to indicate another output table — data preparation and calculation is the same as it is when output is directed to the predictions table.

3.2. Indices to raster files

Three folders containing raster files are potentially involved in this task: input layers, limiting polygons and output files. There are some limitations in using the panel *Indices to raster* in the **Constud** main window. Firstly, the output folder must be accessible as a local folder: a folder physically located in or mapped to the user's computer. Normally you do not have write access to an FTP folder. Secondly, the input folder must be a local folder because all rst-files in the input folder are sequentially transformed to output files by **Constud**. An FTP folder usually contains more files than are needed to transform for a particular mapping task. There is no need to process all map sheets of a data layer when the output map does not exceed the boundaries of one map sheet.

Only three alternative configurations remain — 1) limiting polygons in a local folder, 2) limiting polygons in an FTP folder, and 3) limiting polygons are not used. In the first case, select the *local* option and consider the use of a folder prefix as a cookie-cutter for naming both folders. In the second case, select option *FTP* (Fig 3-7) and define the folder prefix valid only for the data layer of limiting polygons;

the full name of the source data folder must be recorded in the table of data layers *[D_LAYERS].[folder]*.

30. Create a new local folder for input files; name this folder, e.g. C:\soil_10.

Constud transforms all rst-files from the input folder.

31. Copy files 5434.rst, 5434.rdc, 5444.rst, 5444.rdc to the new folder.

These files contain generalised soil units for map sheets 5434 and 5444. Eutrophic peat soils have the numeric code 10.

32. Add a new record to the D_LAYERS table presenting the Estonian 1: 10,000 soil map generalised to 10 categories.

Give the field *folder* the same name you gave to the newly created folder (task 30). *[D_LAYERS].[folder]* = {the folder you created in task 30}.

You can use the following query but don't forget to replace the name *C:\soil_10* in the example with the actual full name of the folder in your computer. The full name of a folder starts with the letter of a computer drive.

```
INSERT INTO [D_LAYERS] ([KID],[p_edge],[folder],[nominal],[void],[fr
om],[until],[bytes],[remarks])
VALUES (5,10, 'C:\soil_10' ,True,0,null,null,1,null);
```

33. Switch off calculation of all existing features.

Only feature 10 is calculated to raster files.

In SQL it looks like:

```
UPDATE [EXPL_VAR] SET [F1] = False;
```

34. Add a new record to the EXPL_VAR table. Connect the new feature to the newly added data layer. Define the feature as the proportion of eutrophic peat soils within 100 m boundaries of the same areal category from layer 1 as at the location of the observation.

Eutrophic peat soils use code 10 in the recently defined data layer number 5. Indicate the binary output folder (*BO_folder*). The output folder should be different from the input folder defined in *D_LAYERS*. The source files will be overwritten if the input and output folders are identical. You can use the following query to add the record to table *EXPL_VAR*. Don't forget to replace the name *C:\soil_10\output* with the actual full name of the folder in your computer. The full name of output folders is

obligatory — the folder prefix is not applied for output folders. If the output folder does not exist then **Constud** will create it.

The folder prefix defined in the main window is applied for input files only for local (not FTP) folders. If a feature is calculated using a pre-classifier located on an FTP server, then the FTP option must be selected from the **Constud** main window (Fig 3-7). Possible combinations of data layers in FTP and local folders are discussed at the beginning of Chapter 3; the main attributes of features are listed in Chapter 2.1.

```
INSERT INTO [EXPL_VAR] (AID, name, KID, index_ID, radius, in_radius,
sample, blocksample, given, precl, precl_layer, BO_folder, F1)
VALUES (10,'proportion of eutrophic peat soils', 5, 1, 100, 0,1,
False,10, True,1, 'C:\soil_10\output', True);
```

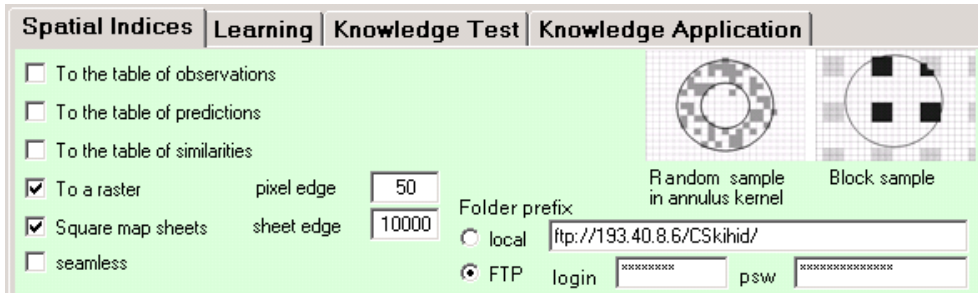


Figure 3-7. **Constud** settings selected for calculating spatial indices to raster files.

35. Launch Constud. Select tab panel 'Spatial Indices' and option 'To raster'. Input the value 50 to the input window pixel edge in order to reduce the number of pixels to be calculated.

If using a layer of pre-classifying polygons from an FTP source, then select the *FTP* radio button and write a folder prefix valid for this FTP server (Fig 3-7). If the *FTP* option is selected then a folder prefix is applied only for the pre-classifier — the full path of input and output files will be read from the database.

Calculation of spatial indices to raster files using **Constud** is still available only for square-shaped files and in non-seamless mode. Use **LSTATS** software (http://www.geo.ut.ee/LSTATS) for calculating the same spatial indices from non-square raster files.

36. To observe or convert the results in Idrisi, add rdc-files to the output rst-files.

Create a new text document (a **Notepad** file, not an **MS Word** document) in the output folder. Name this file *5434.rdc* and copy the contents from here. For *5444.rdc,* the minimum and maximum Y coordinates should be larger by 10,000 m than in *5434.rdc*.

file format.: IDRISI Raster A.1
file title..: 5434
data type...: byte
file type...: binary
columns.....: 250
rows.....: 250
ref. system.: plane
ref. units..: m
unit dist...: 1.0000000
min. X.....: 640000
max. X.....: 650000
min. Y.....: 6430000
max. Y.....: 6440000
pos'n error.: unknown
resolution..: unknown
min. value..: 0
max. value..: 255
display min.: 0
display max.: 255

3.3. Indices from an integer format data layer

All feature values in **Constud** are stored in one-byte format. One byte per channel is a traditional format in most remote sensing data and is also sufficient for categorical coverages (maps of nominal variables). In some cases, the one-byte-per-pixel format is not sufficient. For example, it enables to store only extremely rough elevation models.

Elevation in decimetres stored as short integers (two bytes per pixel) provides reasonable precision for involving land surface elevation data into land cover mapping. Actually, we already applied an integer format data layer in the ninth task. See the record where *KID* = 4 in the *D_LAYERS table* — there you will see *[D_LAYERS].[bytes]* = 2. This means pixel values in this layer are in the two-byte format. Feature number 3 (*relative elevation*) was calculated from this layer. The values of relative elevation were stored in decimetres since no converting was ordered. It was possible since the differences in elevation in the study area are modest.

Calculating the mean elevation from a two-byte data layer would yield the massive maximum possible feature value of 254 (the value 255 is reserved for missing data and undetermined cases). To enable automatic conversion of such results to the one-byte format, the *[EXPL_VAR].[divisor]* field is included in the **Constud** database. The divisor value in this field affects output both to database and to raster files. The calculated values are divided by the divisor. Divisor = 1 is the default value if the

divisor field is empty. A divisor is applied only for data layers that have more than two bytes per pixel.

37. Copy the row where AID = 3 from the EXPL_VAR table. Change the AID field in the copied row to 11 and field [EXPL_VAR].[divisor] to 10. Name this feature as 'Relative elevation in metres'. Switch off calculation of all features except this new feature.

38. Add field 11 to the table of observations (OBSERVATIONS1) and to the estimated values table (ESTIMATED_VALUES_1).

39. Launch Constud to calculate relative elevation in metres to the table of observations and to the table of estimated values.

Compare the values of feature 3 (relative elevation in decimetres) and feature 11 (relative elevation in metres). Remember that a constant of 100 is added to the relative value to store both negative and positive values in byte format.

40. Define a new nominal output feature (AID = 12) in the EXPL_VAR table. Add a corresponding field to the ESTIMATED_VALUES_1 table. Calculate the values of the feature using Constud.

Recall that the basic components of a spatial feature are: the data layer, the index, and the radius for calculating the index. The list of indices available in **Constud** is in the *SP_Indices* table. The *nomlayer* field indicates whether the source data layer is nominal or numerical, the *nomind* field indicates whether the calculated feature is nominal or numerical. As you can see from the table, all indices that result in a nominal feature are one or another modification of the mode.

For a numerical feature, you can use data layer number 4 that was already added and located in folder *ETM-186_019-23-05-2005-B8*. You can also look for other remote sensing data layers in ADSD and register these to *DEP_VAR*. All files in ADSD section *CSkihid* are 10 by 10 km map sheets, and remote sensing data layers are all in one byte format. Following that you can calculate the number of pixels and the length of pixel edge from the file size.

41. Calculate the values of feature AID = 12 using Constud.

Questions

1. According to **Constud** spatial index 115 calculated within 50 m, which are more common among the given observation sites — positive or negative land forms? Coordinates of observation sites are in the database *Constud_tutorial_data.mdb,* you calculated indices to the *OBSERVATIONS1* table.

2. Features 1, 4 and 5 — the proportion of forest within 50 m differ only by the proportion of the sample of pixels in the kernel. Make a scatterplot depicting records in the table of observations (feature 1 on the horizontal and 5 on the vertical axis). Coordinates of observation sites are in the database *Constud_tutorial_data.mdb,* you calculated indices to the *OBSERVATIONS1* table.

3. Feature 7 differs from feature 6 only by application of limiting polygons from layer 1. In how many observed locations are the values of features 6 and 7 different? Coordinates of observation sites are in the database *Constud_tutorial_data.mdb,* you calculated indices to the *OBSERVATIONS1* table.

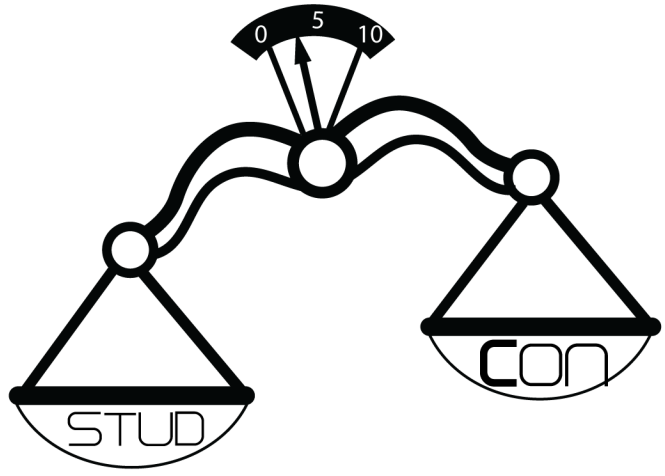
4. What is the average value of feature 6 (standard deviation of pixel values in Landsat image) at locations in which the vicinity is covered only by forest within 50 m and in locations without forest within 50 m? Coordinates of observation sites are in the database *Constud_tutorial_data.mdb,* you calculated indices to the *OBSERVATIONS1* table. The share of forest within 50 m is recorded in field *[1]*.

5. For how many observations is the value of feature 9 (the number of categories within 100 m calculated from a seamless data layer) larger than for feature 8 (number of categories within 100 m in the same map sheet)? Coordinates of observation sites are in the database *Constud_tutorial_data.mdb,* you calculated indices to the *OBSERVATIONS1* table.

6. Lake Pühajärv (Holy Lake) is in the middle of map sheet 5434. What is the calculated value of feature number 10 (the proportion of eutrophic peat soils within 100 m) in this lake and in all other lakes as well as in settlements? How can you explain this value for a proportion?

7. Report the attributes of feature number 12 from *EXP_VAR* and describe the calculated results. You calculated feature number 12 according to task 40.

Weights 4



4.1. Machine learning in Constud

Constud represents an empirical, case-based approach that, instead of generalizations, estimates the predictable values by raw exemplars selected from the learning data. The value of a predictable variable is transferred from the most similar exemplars to the predictable case (observation).

Similarity is estimated between objects (entities, observations, locations, cases) according to features of these objects. The main questions in similarity-based reasoning are: similar in what aspect and similarity to which etalons to rely on? In more detail, which weight must one assign to every descriptive feature and etalon? If an expert decides on the relevance of features and etalons himself/herself then the prediction will probably be subjectively biased. The aim of automated fitting of weights for features and for exemplars is to reduce such bias. Although learning in **Constud** is an automated process, the selection of source data and settings for **Constud** learning are still the user's choice.

> Similarity is estimated between objects according to features of these objects. Objects similar in one aspect may be different in other aspects.

Machine learning in **Constud** is an iterative process for finding the most indicative set of feature weights and exemplar weights. A zero weight means that this particular feature or observation is not needed to calculate predictions. The result of machine learning in **Constud** is a predictive set of weights for features and exemplars.

> The best predictive set consists of weights for features and exemplars.

The order of machine learning subroutines is pre-set in **Constud** (Fig 4-1). The first iterations deal with feature selection, and then feature weighting, selection of exemplars, weighting of exemplars, change of actuality values of features and exemplars and saving the results of the learning iteration follow. The number of iterations for feature weighting and for exemplar weighting is given by the user along with other settings (Chapter 4.2). The machine learning subroutines use the following simple algorithms.

Feature selection. Features are added one by one in order of gradually decreasing weight and using all observations since exemplars are still not selected. The set of features that gave the best fit continues in the feature weighting stage.

Feature weighting. The weights of selected features are increased and decreased

by a small random value; if the goodness-of-fit improves then the new weights are transferred to the next iteration. **Constud** learning is not deterministic but a semi-random process because of the random component in feature weighting.

Calculation of indices (if random samples are not used) and knowledge application in Constud is deterministic; Constud learning is not.

Selection of exemplars. The effect of exclusion of every single observation to prediction fit is compared to the fit in case of observation inclusion. The optimization of weights continues with the option that fits better to all other observations in the learning sample. Selected exemplars continue in the feature weighting stage.

Exemplar weighting. Three options are compared for every observation one by one: 1) increased, 2) decreased and 3) unchanged exemplar weight of the observation. The best predictive option is retained. Exemplar weighting is reiterated at next level of change after all observations have been tested. The change of weight values diminishes twice at every repetition; the initial value is 0.5. More details are given in the description of the setting *Weighting of exemplars* in Chapter 4.2.

The weights of features and exemplars are stored in the knowledge base if a hitherto the best fit has been reached.

Shifting actuality values. The initial actuality value of features and observations is 100 if not set otherwise by the user. The actuality of features and observations that have been selected in a random sample and were useful in prediction is increased proportionally to the weight. The range of actuality values recorded by **Constud** is between 1 and 200. The indicator value of features and exemplars depends on the particular combination they form in a random sample. A feature highly indicative in one set may be useless in a different set of features. The actuality of features and observations that were in the learning sample but were excluded from the best predictive set is decreased. The probability of inclusion of a feature and of an observation to a learning sample is proportional to the particular actuality value.

Actualities form Constud's memory.

The sensitivity of actualities (credulity/incredulity of the system) can be tuned using sliders in **Constud**'s main window (Fig 4-3). When the learning process is set to be more credulous, the actualities are changed faster and more actual features and observations turn out with less iteration.

After approximately ten learning iterations, actualities of features and exemplars are

normalised. In normalizing, the mean actuality of features and exemplars currently in use remains equal to 100.

The mean weight of features is constantly normalised to 1, which means that the sum of weights equals the number of features. The mean weight of exemplars is not normalised.

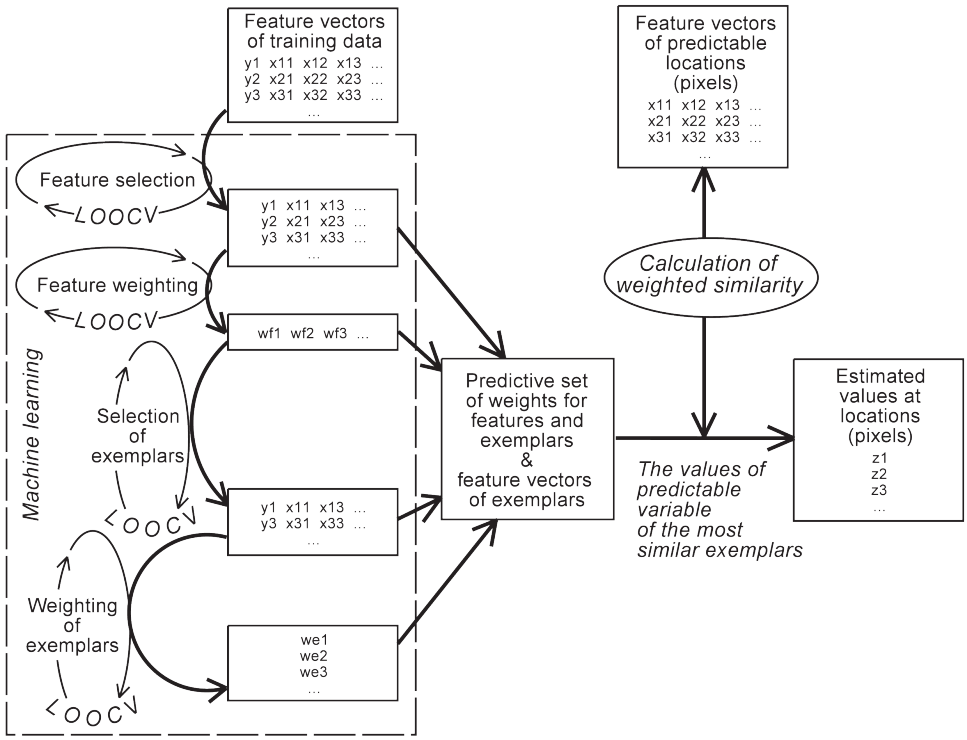


Figure 4-1. Technological schema of machine learning and prediction in **Constud** (modified from *Tamm and Remm, 2009*). **LOOCV** — leave-one-out cross-validation.

4.1.1. Similarity algorithm

Similarity between observations is initially calculated as a partial similarity in single features. The algorithm of partial similarity depends on the type of a particular explanatory feature: nominal or numerical. If the categories of a nominal variable match then the partial similarity is equal to 1; if not (and self-similarities between categories are not defined), then partial similarity equals 0. If the similarity is defined for the particular pair of categories then similarity values from a self-similarity table (registered in the table *Self-Sim-Tables*) will replace the default zero similarity between these categories.

For the numerical feature (f), the difference (D) between its values (T_f and E_f) for an

exemplar (*E*) and a learning instance (*T*) is calculated as:

$$D = \frac{|T_f - E_f|}{V \cdot w_E \cdot w_f},$$

where: V — similarity range in SD, w_E — weight of exemplar E, w_f — weight of feature f.

The weights of features and exemplars are the result of iterative learning; the similarity range V is the level of difference in values of an explanatory variable, exceeding which the observations are considered to be not similar at all in the aspect of this feature. V is measured in standard deviations. The partial similarity (S_f) between an exemplar and an observation regarding feature f is assigned a value of $1 - D$ if $D < 1$; otherwise $S_f = 0$.

The total similarity is calculated as a weighted average of partial similarities, feature and observation weights applied the second time. These weights were also used in calculating partial weights. The double usage of weights regulates both the relative effect of a feature on the total similarity and the extent of the similarity kernel along the feature axis (Fig 4-2).

Zero-weight features, missing data features, features not suitable due to temporal limits and features not applicable for other reasons are excluded from summing up the partial similarities to the total similarity. For example, if soil data for one location are missing, soil types do not affect the total similarity between locations.

For spatial data, a distance correction parameter — *zero distance* that is stored in the **Constud** database as *[DEP_VAR].[0_dist]* is involved for reducing the effect of spatial autocorrelation. This parameter inhibits reciprocal prediction between observations at a close distance. As a result, exemplars which are spatially more dispersed are preferred.

The zero distance is applied through a similarity correction factor, which is calculated within all pairs of locations in a learning sample as the zero distance divided by the distance between the locations. The correction is subtracted from the initial total similarity. The result is compared to the *zero similarity* parameter that can be adjusted in **Constud**'s main window. Similarity values which are less than *zero similarity* will be assigned *zero similarity* value.

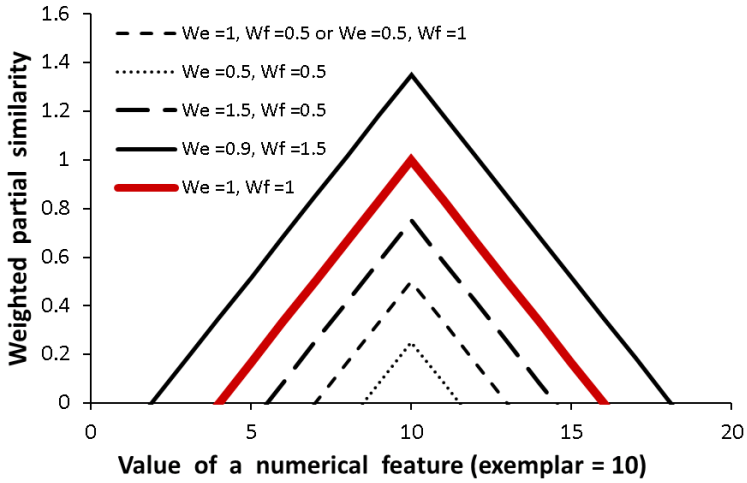


Figure 4-2. The effect of weights on similarity between observations as applied in **Constud**. *We* — weight of the exemplar; *Wf* — weight of the feature, lines depict total similarity depending on the value of this single numerical feature of an observation presuming the exemplar has a value = 10; standard deviation = 3.

4.1.2. Decision making

If only one exemplar is used in a similarity-based estimation then the predicted value is transferred from the exemplar. In most cases, more than one exemplar is used (the *k*-nearest neighbour method — *k-NN*). The *k*-value (number of exemplars used in estimations) is controlled computationally as *the sum of similarity sought for decision*. It is a parameter in **Constud** for which the value can be optimised together with the feature weights. The initial value of *k* has to be input by the user to the field *[DEP_VAR].[sumsimmax]*.

The process of finding the most similar exemplars starts from similarity equal to one. If the total similarity of exemplars above this level is less than *the sum of similarity sought for decision* then the level is lowered by 0.01. The search for the most similar exemplars continues until the necessary sum of similarity is obtained or exceeded or there are no more similar exemplars. Decision making follows after the search is completed. The influence of every selected exemplar to the decision is proportional to its similarity. Relatively small values of the parameter *similarity sought for decision* may yield in higher prediction fit, but on the other hand, **Constud**'s predictions are less sensitive to random noise and occasional extreme values in learning data if the parameter is larger.

4.1.3. Objective function

Goodness-of-fit of the learning results in **Constud** is estimated by leave-one-out cross-validation (LOOCV). LOOCV means that the predicted value for every observation is calculated using all exemplars but leaving this observation out. Objective function is a rule for preferring one predictive set to another. It depends on the kind of dependent variable and is described according to the type of predictable variable in Chapters 4.4–4.8.

4.2. Settings for Constud learning

The **Constud** user has the opportunity to alter the following settings from the *Learning panel* of the *Constud main window* (Fig 4-3).

Spatial data — if selected, then coordinates of observation sites are obligatory and similarity reduction of spatially close observations in learning data is applied. **Constud** is also able to learn non-spatial variables presented in a table as feature vectors.

Fitting the sum of similarity — if selected, then the amount of similarity sought for decision is optimised along with feature weights. If unselected then the value in field *[DEP_VAR].[sumsimmax]* is kept constant.

SD from the table STDEV — if selected, then the standard deviation of numerical features is read from the table *STDEV*. If unselected, then the standard deviation of numerical features is calculated from the sample used in learning iteration. That takes a while, and a learning sample usually does not include all learning data; therefore, the estimated standard deviation is less reliable.

Error messages — if checked, **Constud** displays error messages during learning. This checkbox has influence only on learning and does not suppress error messages when **Constud** is used for other purposes.

Weighting of exemplars — the number of iterations spent for optimizing the weights of exemplars. The possible values are: 0, 1, 2 and 3. Zero means that observations are either selected to or omitted from the set of predictive exemplars and no weighting occurs; the value 1 enables the possible observation weights: 0, 0.5, 1 and 1.5; the value 2 yields in the weights' array: 0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75; and three enables the array: 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1, 1.125, 1.25, 1.5, 1.625, 1.75 and 1.875. During weighting iterations, exemplar weights are changed by one step within the current range of values.

Weighting of features — the number of iterations spent for optimizing the weights of features. Unlike exemplar weighting, feature weights are changed by a random value in the interval −0.5…+0.5.

End if ID of iteration ≥ — exits learning if the ID number of iteration in the log table is larger than the appointed value. This parameter enables to plan and execute experiments that involve multiple comparable learning attempts. Results of machine learning experiments using different data or altered learning parameters should be comparable when the number of learning iterations is the same.

Zero similarity — a similarity level under which the comparable observations are considered to be not similar at all. Normally this parameter should not be modified. The parameter can have values as a single format number. Values ≥ 1 annul all similarity values to zero, negative values include exemplars which differ more in continuous features than would be included normally.

Similarity range in SD — the difference between values of a numerical feature measured in standard deviations exceeding which the observations are considered not to be similar at all regarding this feature. Similarity range affects the width of the kernel along a feature axis (Fig 4-4).

Credulous/cautious while weighting of cases — the track bar value affects the magnitude of change in actuality values for observations. The actuality value of an observation affects the probability of inclusion of this observation to a learning sample during the following learning iterations.

Credulous/cautious while weighting of features — the track bar value affects the magnitude of change in actuality values for features. The actuality value of a feature affects the probability of inclusion of this feature to a learning sample during the following learning iterations. Actuality values reflect the indicator values of features. Actuality values are higher for the features that are useful for estimating the predictable value.

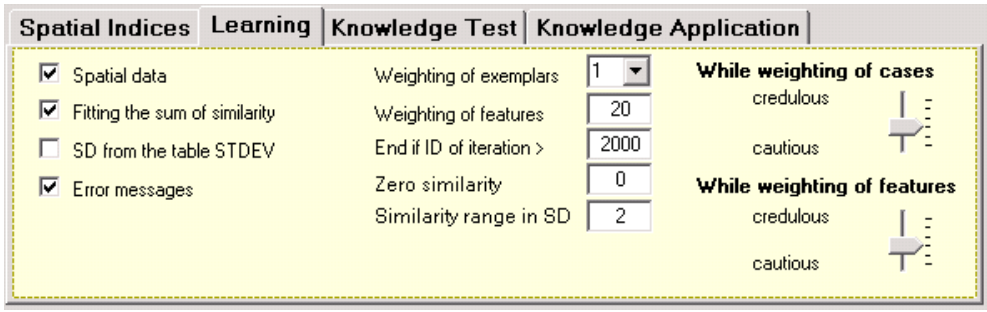


Figure 4-3. Settings in the machine learning panel of **Constud**'s main window.

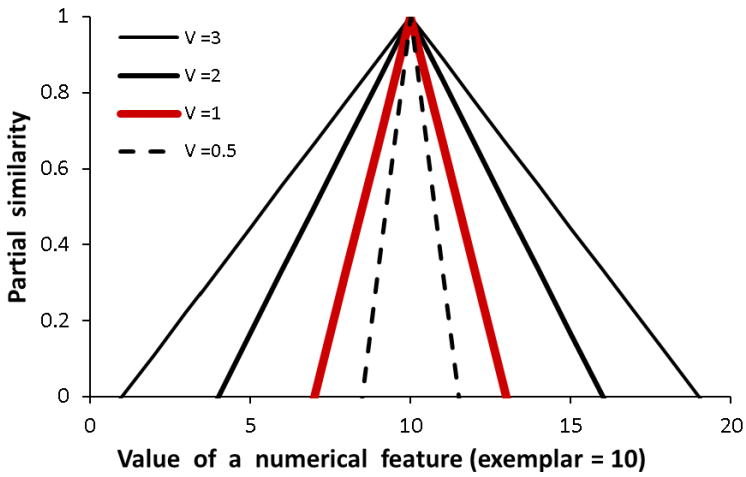


Figure 4-4. The effect of similarity range on calculated similarity value.

Constud learning, even if using exactly the same settings, does not reach the same weights and actuality values while restarted since feature weights are shifted by a random value while learning and the random shift differs each time.

Constud learning involves a random component and is not identically repeatable.

You can interrupt **Constud** learning:

1) by clicking the *x* button in the upper right corner of the process window,

2) from the **Windows Task Manager** (*End process* or *End task*) or

3) by unchecking all rows in *[DEP_VAR].[calc]* (the **Constud** process stops after completing the current iteration).

The results of current learning are not lost since learning results are written to the **Constud** database while the iteration ends. When **Constud** is restarted, learning continues using the last saved weights and actuality values.

The interim result of Constud learning is saved in the end of each learning iteration.

4.3. Pre-classifier of the dependent variable

A pre-classifier feature of the dependent variable limits the set of exemplars, which are used for predictions, by a category of the pre-classifier. The pre-classifier is one of the explanatory variables with its above zero AID number in the field *[DEP_VAR].[precl]*. The pre-classifier of a dependent variable affects both machine learning and calculation of predictions in **Constud**. If pre-classes are applied, then a separate set of exemplars for every category of the pre-classifier is selected. The exemplars are different but the weights for explanatory features are the same for all pre-classes.

The weights for explanatory features are the same for all pre-classes; a separate set of exemplars for every pre-class is selected.

4.4. Learning a Boolean variable

A Boolean (binomial binary) variable has only two possible values; e.g. one and zero, or true and false, or positive and negative cases.

The fit of a Boolean variable is calculated in **Constud** as a modified true skill statistic (TSS): the share of true positive similarity plus the share of true negative similarity minus one. TSS assigns equal weight to both correct negative and correct positive estimations and is similar to the Pearson's coefficient of correlation having an interval of possible values between −1 and +1. The minimum value −1 means that all estimations are false; TSS is equal to zero if half of the positive and half of the negative cases have been estimated correctly. A high level of correct estimations could be obtained by ignoring the less common value of a binomial variable; e.g. if the positive cases form only 1% of the total number of observations, and all cases are estimated as negative, then the share of correct estimations is 99% but the TSS equals zero (*Allouche et al., 2006*).

Traditional TSS uses the share of true positive cases and the share of true negative cases. The modification in **Constud** is that the similarity to positive exemplars and the similarity to negative exemplars are summed up instead of simply counting true positive and true negative predictions. The summed similarity to positive cases and to negative cases are both divided by the total sum of similarity yielding the relative similarity to the category of positive and to the category of negative. The predictable case (observation) is assigned to the category which offers a higher relative similarity.

Fields in the **Constud** database for observed and predicted values of a binary variable

can be either in the binary (*True/False*) or in the one-byte format. Zero and one must be used for coding respectively *true* and *false* for the byte format.

Some data preparation is needed before starting **Constud** learning.

42. Turn off calculation of all previous dependent variable(s) in the field [DEP_VAR].[calc] in the Constud database.

43. Add a dependent variable to the Constud database for learning and handling the presence/absence of Epipactis palustris.

Set the size of the learning sample to 100, validation sample to 500, the number of features to 6 and zero-distance to 100. A Boolean variable is coded as: *[DEP_VAR].[ftype]* = 3 (Appendix 2.1 and Chapter 2.2.1). You can copy-paste the only existing row so far in *DEP_VAR* and change it or use the following query.

```
INSERT INTO [DEP_VAR] ([calc], [FID], [given], [log_ID],[precl],
[name], [ftype], [no_part_feature], [sumsimmax], [dataquery],
 [t_sample], [v_sample], [no_features], [0_dist]) VALUES
(True,2,0,0,0, 'presence/absence of E.palustris', 3, 0, 3,
'OBSERVATIONS1', 100,500,6,100);
```

44. Add fields F2 and AF2 to the EXPL_VAR table corresponding to the dependent variable FID = 2.

You can use either button *Fields F... and AF...* in the *Create tables* window or copy the existing *F1* and *AF1* fields or use the following query. The default value for *AF* can be either *Null* or 100.

```
ALTER TABLE [EXPL_VAR] ADD [F2] bit NULL, [AF2] byte NULL;
```

45. Add the LOG_F2 table corresponding to the dependent variable FID = 2.

We suggest using the function *Create tables* in **Constud**. This function produces missing log tables for all dependent variables listed in the *DEP_VAR* table.

46. Turn on application of all features for learning the dependent variable 2 in the field [EXPL_VAR].[F2] where AID is not 10.

There is no need to change the field *[EXPL_VAR].[F1]*. Feature number 10 must be switched off because it is not in the table of observations. Observation sites of *E. palustris* and features of the sites are already in the *OBSERVATIONS1* table.

```
UPDATE [EXPL_VAR] SET [F2] = True WHERE AID<>10;
```

Only features which are actually present in the learning data can be switched on in [EXPL_VAR].[F...].

47. Now and always before you start machine learning in Constud — check the database objects required for learning.

Inspect:

1) the selected variable(s) in the *DEP_VAR* table;

2) the presence of fields *F...* and *AF...* in *EXPL_VAR* corresponding to the selected dependent variable(s);

3) at least some features selected in the field(s) *[EXPL_VAR].[F...]*;

4) fields *VID*, *F*, *pred*, *actuality*, *sim*, *w*, *from*, *until* and fields corresponding to AID numbers of the selected features in the table *EXPL_VAR*, filled with values in the table or view of observations;

5) log tables *LOG_F...* corresponding to the dependent variables and at least one row in every log table (Fig 4-5);

6) close all objects in the **Constud** database which are open for design.

The *Validate tables* button in the **Constud**'s main window (Fig 2-2) helps to check the formal data structure in the **Constud** database but keep in mind that the software is not able to understand the meaning of codes and correctness of observation data.

Check before Constud learning:
1) selected dependent variables;
2) fields F and AF in EXPL_VAR;
3) fields VID, F, pred, actuality, sim, w, from, until and fields for explanatory features in the learning data;
4) log tables.

48. Launch Constud and select the learning panel. Keep default settings (Fig 4-3) except the maximum number of iterations (End if ID of iteration >), for which 200 is enough for a tutorial exercise.

200 learning iterations using about 100 observations and about 6 features in a learning sample takes less than 10 minutes on a modern PC. About 1000–2000

learning iterations are enough for most research experiments. Use parallel learning to duplicate databases rather than increase the number of iterations by some thousands.

You can view the learning progress in the log table that can be open while learning. The learning parameters that are stored in the **Constud** database can also be altered while learning.

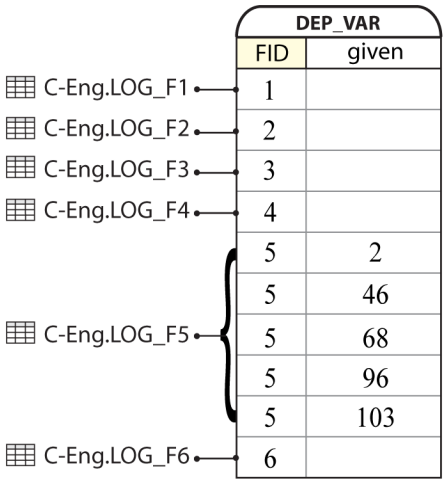


Figure 4-5. Connection between log tables and records in the *DEP_VAR* table.

49. Check the LOG_F2 table and find the record of the learning iteration which best fits the validation sample.

Notice that the ID number of this record was written to *[DEP_VAR].[log_ID]*. The other recorded learning results are: goodness-of-fit in the learning sample — *[LOG_F2].[t_fit]*, goodness-of-fit in the validation sample — *[LOG_F2].[v_fit]*, number of exemplars — *[LOG_F2].[etn]*, size of the learning sample — *[LOG_F2].[sample]*, weights of features — *[LOG_F2].[F_weights]*, actualities of features — *[EXPL_VAR].[AF2]*, actualities of observations — *[OBSERVATIONS1].[actuality]*, weights of exemplars in the best predictive set — *[OBSERVATIONS1].[w]*, predicted values for observations in the validation sample — *[OBSERVATIONS1].[pred]*, similarity to exemplars for the predicted observations — *[OBSERVATIONS1].[sim]*. Feature weights are in the log table as: <*feature AID*> <*feature weight*> <*feature AID*> <*feature weight*>, etc.

The weights and actualities are recorded in the learning sample, predicted values and similarities are recorded in the validation sample.

The following issues are also worthy of attention.

1. The mean value of recorded actualities is close to 100 since actualities are normalised.
2. The sample size does not match the predefined value exactly because the sampling algorithm in **Constud** recalculates the intended sample size to the probability of inclusion and then applies the probability to every observation; the resulting approximate sample size is not corrected in the given intended value.
3. The weights and actualities are recorded in the learning sample; predicted values and similarities are recorded in the validation sample.
4. Records having higher actuality values do not match the records that have higher weights — actualities of both features and cases are cumulative results of the learning process; weights are overwritten every time a hitherto best goodness-of-fit is reached.

Actualities are cumulative indicators; weights represent just one predictive set.

4.5. Multinomial variable

The multinomial variable is a nominal (categorical) variable that has more than two possible values recorded as class codes. The goodness-of-fit of a multinomial variable is compared by the index of classification agreement (kappa coefficient) (*Congalton and Green, 1999*). Theoretically, the values of the kappa coefficient can range from −1 to +1, zero being the expected value in case of random assignment of cases to categories.

Results of a detailed vegetation mapping experiment containing 111 mapping units have been prepared in the *Plant_cover* table in the *Constud_tutorial_data.mdb* as an example dataset of a multinomial variable. The observation records have been prepared to match **Constud** requirements. There is no need to define and calculate explanatory features for this dataset as these data have already been used in research and some features of observation locations are already included.

You only have to:

1) add a row to the table of dependent variables,

2) add fields *F…* and *AF…* corresponding to the selected dependent variable(s),

3) add feature descriptions and select these in *EXPL_VAR* and

4) add a log table corresponding to the new dependent variable.

Details of these four tasks are given below.

50. Import the table containing plant cover data 'Plant_cover' in Constud_tutorial_data.mdb to your Constud database.

51. Add definitions of features 13–50 from the Additional_features table in Constud_tutorial_data.mdb to the EXPL_VAR table in your Constud database.

Actually, we could use any nominal and any numerical index since we are not going to calculate values of the features. Index ID is needed in **Constud** learning only to define the type (nominal/numerical) of a feature (Chapter 2.1).

52. Add definitions for additional data layers 12–33 from the Additional_layers table in Constud_tutorial_data.mdb to the D_LAYERS table in your Constud database.

Actually we could use any data layer that does not have time restrictions since we are not going to calculate feature values this time. Data layers are needed in **Constud** learning only to define temporal validation restrictions of features (Chapter 2.1); the other attributes of data layers were essential for the calculation of spatial indices. The use of universal nominal and universal numerical data layers for the learning of non-spatial variables is demonstrated in Chapter 4.8.

53. Create a view that selects all fields from the Plant_cover table and the records where VID < 301.

Name this view *Plant_cover_learning*. Later we will ask you to create another view involving condition VID > 301 from the same table as a validation sample. In an **SQL Server**, queries are called views.

```
SELECT * FROM [Plant_cover] WHERE VID<301;
```

54. Add a row defined in the following query to the DEP_VAR table.

Check whether the field *dataquery* matches the view created in task 53. Note that

[DEP_VAR].[ftype] = 0 marks a multinomial variable (Appendix 2.1 and Chapter 2.2.1), feature 13 serves as a pre-classifier for the dependent variable, the sample size of both the learning and validation sample is 150, and finally, the data source is not the *Plant_cover* table.

```
INSERT INTO [DEP_VAR]
([calc],[FID],[given],[log_ID], [precl], [name], [ftype],
[no_part_feature], [sumsimmax], [dataquery], [t_sample],
[v_sample], [no_features], [0_dist])
VALUES
(True,3,0,0,13,'111 plant cover units',0,0,2,
'Plant_cover_learning', 150,150,10,100);
```

55. Turn off calculation of the previous dependent variables in the field [DEP_VAR].[calc].

56. Add fields F3 and AF3 to the EXPL_VAR table.

Follow task 44 as an example. Check that the numerical parts of the fields match the FID number of the dependent multinomial variable.

57. Turn [EXPL_VAR].[F3] to True where AID > 12 and to False where AID < 13.

The SQL commands look like:

```
UPDATE [EXPL_VAR] SET [F3] = True WHERE [AID]>12;

UPDATE [EXPL_VAR] SET [F3] = False WHERE [AID]<13;
```

58. Add a log table corresponding to the dependent multinomial variable.

We suggest using the *Create tables* function in **Constud**. This function produces missing log tables for all dependent variables listed in *DEP_VAR*.

59. Check consistency of data structures listed in task 47, then launch Constud learning. Keep default settings (Fig 4-3) except for the maximum number of iterations (End if ID >), for which 200 suffices for a tutorial exercise.

The allocation of learning results in tables of a **Constud** database is listed in task 49. The only difference of a multinomial variable in **Constud** compared to a Boolean variable is the byte format of the dependent variable and its predicted values.

4.6. Numerical variable

Numerical variable (*ftype* 2) is handled as a continuous, four-byte real number. The same type and algorithms are applicable also for variables which originally have

one byte or integer format. Goodness-of-fit of a numerical variable is characterised by the root mean square error (RMSE), where the number of features used for prediction is subtracted from the number of cases to estimate the degrees of freedom in the predictive set.

Precipitation data from the Baltic countries have been prepared in the *Baltic_precipitation* table in the *Constud_tutorial_data.mdb* database as an example dataset of a numerical variable. The fields *VID*, *X*, *Y*, *From*, *Until*, *Station* (the name of observation station), *F* (mean annual amount of precipitation in mm), *pred*, *sim*, *actuality*, *w* for storing learning results; and numerical fields for feature values are all included.

60. Import or copy the Baltic_precipitation table in Constud_tutorial_data.mdb to your Constud database.

This table contains data on the mean annual precipitation in the Baltic countries and a few features describing the location and vicinity of each meteorological station.

61. Copy definitions of features 51–57 from the Additional_features table in Constud_tutorial_data.mdb to the EXPL_VAR table in your Constud database.

These numeric features are:

51 — Lambert-Est west-east-coordinate divided by 5000 (for transforming it to byte format).

52 — Lambert-Est south-north-coordinate divided by 10000 (for transforming it to byte format).

53 — mean annual precipitation in neighbouring stations within 75 km (expressed in cm because feature values must be in byte format in **Constud**).

54 — share of water bodies within a 10 km neighbourhood.

55 — share of forest within a 10 km neighbourhood.

56 — distance to the sea in kilometres.

57 — land elevation in metres.

The only relevant attributes of these features are *KID* and *index_ID* since values of the features have already been calculated (Chapter 2.1).

62. Add definitions for universal nominal (KID = 10) and universal numerical (KID = 11) data layers to the D_LAYERS table.

The features added in a previous task are connected to the universal numerical layer. The universal nominal layer will be used later. These data layers are a universal reference for previously calculated features and non-spatial features. A reference to a data layer is required for every feature in **Constud** (Chapter 1.2).

You can copy-paste records of these layers from the table *Additional_layers* in *Constud_tutorial_data.mdb* or use the following queries.

```
INSERT INTO [D_LAYERS] ([KID],[nominal],[folder])
VALUES (10,True,'universal nominal');

INSERT INTO [D_LAYERS] ([KID],[nominal],[folder])
VALUES (11,False,'universal numerical');
```

63. Turn off calculation of the previous dependent variable(s) in the field [DEP_VAR].[calc].

64. Add a row defined in the following query to the DEP_VAR table.

Check whether the field *dataquery* matches the table imported in the previous task. Notice that *[DEP_VAR].[ftype]* = 2 marks a numerical variable (see also Appendix 2.1 and Chapter 2.2.1) and that both the learning and validation sample involve all 123 observation sites.

```
INSERT INTO [DEP_VAR]
([calc],[FID],[given],[log_ID],[precl],[name],[ftype],
[no_part_feature],[sumsimmax],[dataquery],[t_sample],
[v_sample],[no_features],[0_dist])
VALUES (True,4,0,0,0,'Baltic precipitation',2,0,3,
'Baltic_precipitation',123,123,6,100);
```

65. Add fields F4 and AF4 to the EXPL_VAR table and turn [EXPL_VAR].[F4] = True where AID > 50.

Other records in this field are = *False* by default. Follow task 44 as an example. Check that the numerical parts of the fields match the FID number of the dependent multinomial variable.

66. Add a log table corresponding to this dependent variable.

Follow task 45 as an example.

67. Launch Constud learning. Keep default settings (Fig 4-3) except for the maximum number of iterations (End if ID >), for which 200 is enough for a tutorial exercise.

The results of **Constud** learning are recorded in fields: *AF*, *w*, *actuality*, *sim*, *pred*, *t_fit*, *v_fit*, *F_weights* etc., as in the previous variables. Notice that goodness-of-fit of learning results is recorded as the root mean squared errors in case of a numerical variable — the less the mean value of errors, the better the result.

4.7. Single category of a multinomial variable

Constud can learn to recognize of a single category against all other categories of the same nominal variable. The hitherto best set of weights for features and for exemplars can be selected separately for every category of a nominal variable.

Since recognition of a single category against all another categories is a task that entails splitting observations into two groups, this kind of variable is computationally treated as a binomial in **Constud**. Therefore, the same modified TSS as for a Boolean variable (Chapter 4.4) serves as the objective function.

Although we shall reuse the same plant cover dataset prepared in Chapter 4.5, the same data mounting operations have to be done before the learning begins.

68. Duplicate fields: pred, sim, w and actuality in the Plant_cover table in query design view, assign the corresponding names: pred1, sim1, actuality1 and w1 to the copied fields.

We shall use these fields in the next SQL query. The format of field *w1* must be a single precision real number (*single* in **Access**, *real* in **SQL Server**); the other three fields must be in byte format.

69. Create a view named Plant_cover_learning1 that reads the first 300 records from the Plant_cover table and renames fields: pred1, sim1, actuality1 and w1; assigning names that correspond to the Constud data structure: pred, sim, actuality and w.

You can create strictly fixed names for the database objects required for **Constud** as aliases by renaming the original names in queries or database views. Here we have an example of how to use the same source data for learning different dependent variables. Fields *pred*, *sim*, *actuality* and *w* are needed for both variables while learning. Values will be overwritten if the same fields were used for both variables. Therefore, we created fields: *pred1*, *sim1*, *actuality1* and *w1*. We have to rename the duplicate fields because **Constud** software strictly looks for fields named *pred*, *sim*, *actuality* and *w*.

For an **Access** database, the query should be written as the example below and

named *Plant_cover_learning1*.

```
SELECT [VID],[F],[X],[Y],[pred1] AS pred,[sim1] AS sim,[w1] AS w,
[actuality1] AS actuality, [From], [Until],[13],[14],[15],[16],[17],
[18],[19],[20],[21],[22],[23],[24],[25],[26],[27],[28],[29], [30],
[31],[32],[33],[34],[35],[36],[37],[38],[39],[40],[41],[42],[43],
[44],[45],[46],[47],[48],[49],[50]
FROM [Plant_cover] WHERE [VID]<301;
```

In **SQL Server**, the command starts as:

```
Create view [Schema_name].[Plant_cover_learning1] AS SELECT …
```

Replace `Schema_name` with the actual name of the schema you are working in.

Database objects can be renamed

70. Turn off calculation of the previous dependent variable(s) in [DEP_VAR].[calc].

71. Add a row for every category you intend to learn separately in the DEP_VAR table.

Let's take five relatively common and well-defined categories this time: boreal pine wood (code 2), birch swamp wood (code 46), boreo-nemoral grassland (code 68), fallow (code 96) and ploughed field (code 103).

The SQL query for inserting a record to *DEP_VAR* for learning the boreal pine woods category follows below. Note that the option *ftype* = 4 denotes learning of a single category against others.

```
INSERT INTO [DEP_VAR]
([calc],[FID],[given],[log_ID],[precl], [name], [ftype],
[sumsimmax], [dataquery], [t_sample], [v_sample],[no_features],
[0_dist])
VALUES
(True,5,2,0,13,'boreal pine wood', 4,2,'Plant_cover_learning1',
150,150,10,100);
```

Before adding records for learning the other categories, change the code and the name of the given category to the code and name of the category you are adding. As a result, there should be five rows with *FID* = 5 in the *DEP_VAR* table. Each of these rows differs from the others within the *FID* = 5 series only by the given code that matches the code of the plant cover category, which is what we intend to learn to recognize using remote sensing data.

72. Add fields [F5_…] and [AF5_…] to the EXPL_VAR table.

Note that every category requires these fields separately because features and the actualities used to recognize each category are independent from features and actuality values selected for other categories. You can use either the *Fields F... and AF...* buttons in the *Create tables* window or copy and adjust the existing fields by adding '_' and the code of the category to field names *F...* and *AF...*; or you can use the following SQL commands.

```
ALTER TABLE [EXPL_VAR] ADD [F5_2] bit NULL, [AF5_2] byte NULL;

ALTER TABLE [EXPL_VAR] ADD [F5_46] bit NULL, [AF5_46] byte NULL;

ALTER TABLE [EXPL_VAR] ADD [F5_68] bit NULL, [AF5_68] byte NULL;

ALTER TABLE [EXPL_VAR] ADD [F5_96] bit NULL, [AF5_96] byte NULL;

ALTER TABLE [EXPL_VAR] ADD [F5_103] bit NULL, [AF5_103] byte NULL;
```

73. Add the LOG_F5 table that corresponds to the dependent variable FID = 5

We suggest using the *Create tables* function in **Constud**. This function creates missing log tables for all dependent variables listed in the *DEP_VAR* table. A single log table is common to all categories of the same dependent variable, as the table contains a field to record the given category.

74. Turn on application of explanatory variables on F-fields of the current dependent variable 5 in EXPL_VAR where AID > 13 and AID < 51.

```
UPDATE [EXPL_VAR] SET [F5_2]=True,[F5_46]=True, [F5_68]=True,
[F5_96]=True,[F5_103]=True
WHERE [AID]>12 and [AID]<51;
```

Notice that feature 13 is used as the pre-classifier of the dependent variable and the actuality of this feature is not learned. The value *True* in the *[EXPL_VAR].[F...]* field on the line of a pre-classifier is not necessary, although it does not harm learning.

75. Keep default settings (Fig 4-3) and launch Constud learning.

This task takes a longer time than the previous **Constud** learnings. The default total of 1000 iterations will be shared between learning of the five selected categories — that makes 200 iterations for learning each category. Notice that you can switch on and off learning of any category from the field *[DEP_VAR].[calc]* while learning. Note that the adequacy of recognition improves much faster during the first iterations than it does later (Fig 4-5). An increase in the number of iterations over some hundreds usually does not improve prediction fit significantly.

The allocation of learning results in tables of a **Constud** database is listed at task 49. There are some traits specific for learning results of single categories of a nominal variable.

1. Every category of the same nominal variable has a different iteration number recorded in *[DEP_VAR].[log_ID]* corresponding to the best learning result for this particular category.

2. The log table contains separate records for every learned category, the codes of which are located in the field *given*.

3. The number of exemplars *[etn]* counts only exemplars of the given category and not of alternative categories.

4. Feature weights and exemplar weights are optimised only for recognition of the given category.

5. Actualities of features are shifted and recorded separately for every category.

6. An actuality value has been recorded for records that were included in a learning sample at least once.

7. A predicted value was recorded only when the predicted category was the given category.

8. Similarity values are always equal to or larger than 50. Less similarity means the case is more similar to other categories so the predicted value and similarity are not stored because recognition of only the given category was learned during that particular iteration.

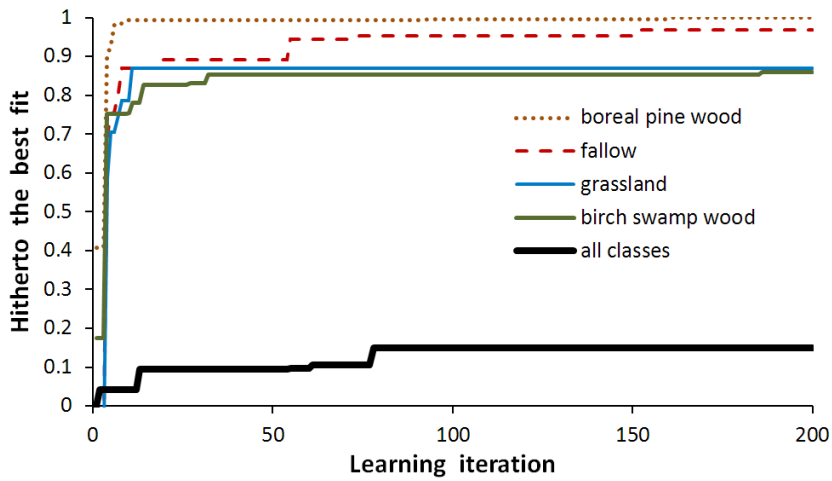


Figure 4-6. Learning curves of separate plant cover categories.

4.8. Non-spatial data

Non-spatial data do not involve spatial coordinates in observation records, such as the results of a questioning coded to numerical one-byte format. The descriptive features of non-spatial records cannot be calculated from data layers using **Constud** — the byte format feature values must be obtained from elsewhere.

> The descriptive features of non-spatial records cannot be calculated from data layers.

There is a significant difference in the **Constud** database mounted for learning and predicting a non-spatial variable — the feature type (nominal or numerical) is derived from the data layer type, not from index type as was the case with spatial data (recall Chapter 2.1). A less significant difference is the unnecessity of tables *SP_Indices* and *Map_Sheets* for learning and predicting a non-spatial variable.

The **Constud** tutorial database contains an *Enterobiasis* table that contains results from a study on enterobiasis occurrence among nursery school children in southeast Estonia (*Remm, 2005*). Enterobiasis is contamination with pinworm (*Enterobius vermicularis*). Every record in the table represents a child. Field *F* in the table marks an enterobius positive or negative person, the descriptive features of every child selected to this example dataset are: *[60]* — age, *[61]* — gender, *[62]* — the number of children in the family, *[63]* — equal-age nursery group (versus children of different ages in the nursery group), *[64]* — age range in the nursery group, *[65]* — geographical region (district), and *[66]* — settlement type. Features 61, 65 and 66 are nominal; the others are numerical. In the following exercise we try to guess using these seven features whether a child is enterobius positive or negative (contaminated or not). We are also going to compare the significance of these features as factors of possible infection risk. Predicting infection risk at an individual level is rather innovative in epidemiology. Usually risk and occurrence are both estimated and modelled at the population level.

For learning of data that lack spatial coordinates, uncheck the checkbox labelled *Spatial data*. Before you can start **Constud** learning, you have to complete the following data mounting tasks.

76. Import or copy the Enterobiasis table in Constud_tutorial_data.mdb to your Constud database.

77. Turn off calculation of all previous dependent variable(s) in the [DEP_VAR]. [calc] field.

78. Add a dependent variable to the Constud database for learning and predicting enterobius positive/negative cases as a Boolean dependent variable.

Recall that Boolean variables are coded as 3 in *[DEP_VAR].[ftype]*.

```
INSERT INTO [DEP_VAR] ([calc],[FID],[name],[ftype], [sumsimmax],
[dataquery], [t_sample], [v_sample], [no_features])
VALUES (True,6,'enterobiasis',3,3,'Enterobiasis', 200,500,6);
```

79. Add definitions of features 60...66 from the Additional_features table in Constud_tutorial_data.mdb to the EXPL_VAR table in your Constud database.

You can use copy-paste if using **Access**. The only relevant attribute of these features, in addition to AID, is KID, since values of the features have already been calculated (see also Chapter 2.1).

80. Add fields F6 and AF6 to the EXPL_VAR table corresponding to the dependent variable FID = 6.

You can use either the *Fields F... and AF...* button in the *Create tables* window, or copy existing fields *F1* and *AF1;* or you can use the following query. The default value of *AF* can be either *Null* or 100; the last option does not function in this SQL command in all database systems.

```
ALTER TABLE [EXPL_VAR] ADD [F6] bit NULL, [AF6] byte NULL;
```

81. Update [EXPL_VAR].[F6] = True where AID > 59 and = False where AID <67.

82. Add the LOG_F6 table corresponding to the dependent variable FID = 6.

We suggest using the *Create tables* function in **Constud**. This function creates missing log tables for all dependent variables listed in *DEP_VAR*.

83. Uncheck the option box 'Spatial data', keep the default settings (Fig 4-3) and start Constud learning.

Wait while the results of a few first learning iterations have been recorded, then you can leave the computer learning alone for a longer time. You can reduce the number of learning iterations below the default value of 1000 if you are in a hurry. About 200–500 iterations are sufficient for the next task. Results of this learning are analogous to a spatial binomial variable (Chapter 4.4).

4.9. Multi-dimensional variable

For a multi-dimensional numerical variable, the goodness-of-fit is measured by the mean extent of co-incidence in all dimensions of this variable. The option to use a multi-dimensional variable has not yet been fully implemented and is still not available in **Constud**.

Questions

1. What is the best possible goodness-of-fit for a binary variable and for a numerical variable in **Constud**?

2. Which type of variable has usually duplicate values in the field *[DEP_VAR].[FID]* in the **Constud** database?

3. Why is the goodness-of-fit better in learning samples than in validation samples? Is a different result possible?

4. Should the actuality values be higher for features that have larger weights in the best predictive set?

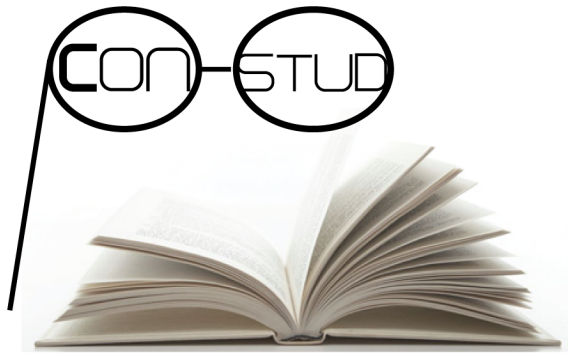
5. Both weights and actualities are stored as a result of **Constud** learning. Artificial intelligence systems have memory. In what sense do actualities comprise the memory of the **Constud** system but the weights do not?

6. Which goodness-of-fit value did your learning experiments reach in **Constud** for the presence/absence of *Epipactis palustris* (Chapter 4.4), for precipitation data (Chapter 4.5) and for plant cover (Chapter 4.6)?

7. The plant cover unit is not predicted for observations VID = 173 and VID = 205 because of feature AID = 13. Explain the reason for this.

8. Create a graph depicting the learning curve of enterobiasis presence/absence in your experiment. An example is given in Fig 4-6.

Post processing 5



There are only three options in the panel: *Knowledge Test*: *Validation, Reselection of exemplars* and *Log table browsing and parsing* (Fig 5-1). The log table browsing tool supports examination of learning results and saving feature weights to a parse table where each of the features has recorded weights either in one row or in one column. The *Validation* option is needed: firstly — to correct validation fit according to all learning data and to amend predicted values resulting from samples; and secondly — if the learning data have been updated. The third option: *Reselection of exemplars*, checks every case one by one, whether a change of weight would improve goodness-of-fit of the predictions.

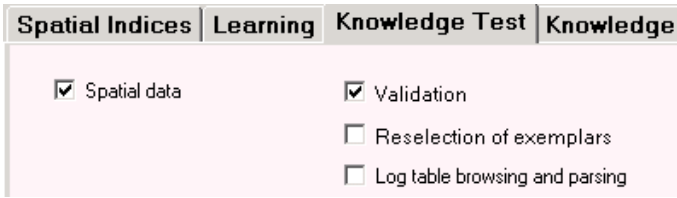


Figure 5-1. Knowledge test options in **Constud**.

Don't forget to check/uncheck the option: Spatial data before validation.

5.1. Log table browsing and parsing

This tool enables to explore log tables of dependent variables and to parse and save these tables to a text format file. Parsing in **Constud** means resolving the string of feature weights to a table where each of the features has a separate row or column.

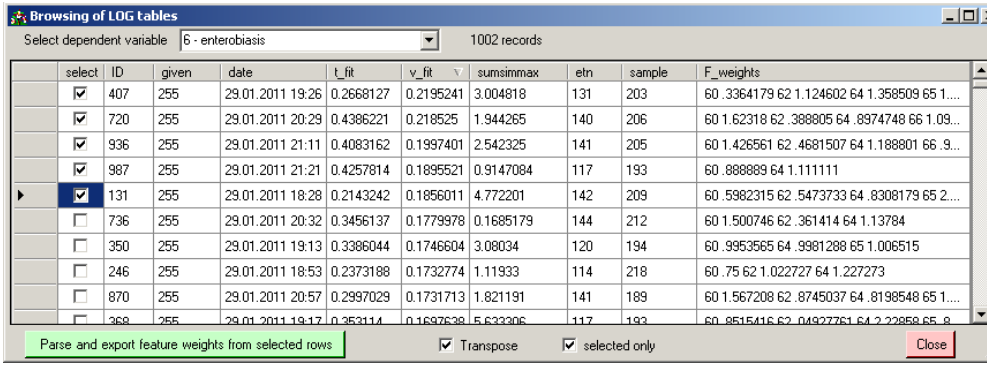
84. Start Constud, select the same database you have used in previous tasks, select the Knowledge Test tab panel and Log table browsing and parsing from there (Fig 5-1). Press Continue.

The window for browsing log tables opens (Fig 5-2).

85. Select dependent variable number 6 from the upper part of the log table browsing window (Fig 5-2).

The names of dependent variables and their FID numbers are read from the *DEP_VAR* table in the **Constud** database. The records of learning results are read from the log table of the selected dependent variable. The records in the browser can be ordered according to the selected column. The column of feature weights and the whole form can be extended.

The weights of features are recorded as: <AID> <weight> <AID> <weight> etc.



select	ID	given	date	t_fit	v_fit	sumsimmax	etn	sample	F_weights
☑	407	255	29.01.2011 19:26	0.2668127	0.2195241	3.004818	131	203	60 .3364179 62 1.124602 64 1.358509 65 1....
☑	720	255	29.01.2011 20:29	0.4386221	0.218525	1.944265	140	206	60 1.62318 62 .388805 64 .8974748 66 1.09...
☑	936	255	29.01.2011 21:11	0.4083162	0.1997401	2.542325	141	205	60 1.426561 62 .4681507 64 1.188801 66 .9...
☑	987	255	29.01.2011 21:21	0.4257814	0.1895521	0.9147084	117	193	60 .888889 64 1.111111
☑	131	255	29.01.2011 18:28	0.2143242	0.1856011	4.772201	142	209	60 .5982315 62 .5473733 64 .8308179 65 2....
☐	736	255	29.01.2011 20:32	0.3456137	0.1779978	0.1685179	144	212	60 1.500746 62 .361414 64 1.13784
☐	350	255	29.01.2011 19:13	0.3386044	0.1746604	3.08034	120	194	60 .9953565 64 .9981288 65 1.006515
☐	246	255	29.01.2011 18:53	0.2373188	0.1732774	1.11933	114	218	60 .75 62 1.022727 64 1.227273
☐	870	255	29.01.2011 20:57	0.2997029	0.1731713	1.821191	141	189	60 1.567208 62 .8745037 64 .8198548 65 1....

Figure 5-2. Constud window for browsing log tables. Five rows of the best iterations according to validation fit are selected in the first column.

86. Order the log table browser according to validation fit and select the best five learning results (Fig 5-2).

87. Try to parse the strings of feature weights to rows (transpose) and to columns.

You can import the exported table of feature weights to **Excel**, where it is easy to calculate summary statistics along rows and columns. Notice that the order of features according to their mean weight is not the same as according to their actuality. Also be aware that for a numerical variable, a smaller value means a better fit because the recorded number expresses the mean error — the root mean squared error. For other types of variables a larger number expresses a better fit between predictions and the observed data.

5.2. Recalculation of validation fit and predicted values

The *Validation fit* checkbox is designed to get validation fit and predicted values for all learning data (Fig 5-1).

A value for the hitherto best fit obtained during machine learning and recorded in the log table and in the *[DEP_VAR]* table is calculated from a validation sample. The size of validation samples is controlled by the parameter *[DEP_VAR].[v_sample]*. In machine learning, if the value in the *v_sample* field is less than the total number of learning observations, the validation fit is calculated from a part of the data. During machine learning, the predicted values are recorded only for observations that fell into a validation sample; that is why validation fit is valid only for this validation sample and not for all learning data. When a subsequent hitherto best result is reached, the predicted values are calculated using different feature weights and

exemplar weights, and are recorded for members of another validation sample. Therefore, the set of predicted values stored to the learning data is a set derived from different samples using different weights for features and cases.

If validation sample < learning data then the predicted values recorded during Constud learning derive from different validation samples

Recalculation of the recorded validation fit is necessary when the set of exemplars and/or features has been modified (cases added or removed, feature choice or feature weights being modified), or when learning or validation samples were used. The recalculation of validation fit and predicted values does not change anything when the volume of learning sample and the volume of validation sample are defined to be at least the same size as the total number of observations in the learning dataset.

If the database object containing learning records of the current dependent variable includes a field named *[exemplars_used]* then the VID numbers and the similarity of exemplars used to predict the dependent variable are saved to this field for each record. Analysing these strings reveals the origin of every predicted value and helps to explain false predictions. Usually, the false predictions are resulted either from atypical or from erroneous cases in learning data. Machine learning is not responsible for representativeness and correctness of learning data.

Constud is not responsible for representativeness and correctness of learning data.

88. Make a copy of the table containing Baltic precipitation data and of the log table row where the best learning result of Baltic precipitation is stored.

The name of the log table containing a report from the learning of Baltic precipitation data contains the FID number of the variable *Baltic precipiation* recorded in *DEP_VAR* (Fig 4-5). You can copy the whole log table as well. Copying is needed because the calculation of validation fit in **Constud** results in overwriting of the learning results of the best iteration. The copy can be anywhere in any format, it is needed only for comparing records stored before and after recalculating validation fit.

89. Update the field [DEP_VAR].[calc] = True in the row where the dependent variable is Baltic precipitation. Update [DEP_VAR].[calc] = False in all other rows.

90. Start the calculation of validation fit. Compare the new validation fit and the feature weights with the initial ones in the copied log table.

Only the recording time changed because all 123 observation stations were included to every validation sample while learning.

91. Compare the weights of exemplars and the predicted values in the two versions of learning data table.

It is convenient to arrange the comparable fields side by side by using the following SQL query.

```
SELECT [Baltic_precipitation].[pred], [Baltic_precipitation_copy].
[pred], [Baltic_precipitation].[w], [Baltic_precipitation_copy].[w]
FROM [Baltic_precipitation_copy] INNER JOIN [Baltic_precipitation]
ON [Baltic_precipitation_copy].[VID] = [Baltic_precipitation].[VID];
```

Separate calculation of the validation fit does not change learning results if the size of the validation sample while learning was large enough to include all records.

92. Add the records of five observation stations from the Baltic_precip_plus table in Constud_tutorial_data.mdb to the Baltic_precipitation table in your Constud database.

Addition of observation records after learning a variable is a typical situation in practice. We need to know how much the added knowledge altered the expected reliability of our similarity-based prediction system.

93. Start the calculation of validation fit in Constud. Compare the calculated fit with the previous fit; compare also feature weights of the best predictive set in the log table.

This time the validation fit of the best learning result should change.

94. Compare the weights of exemplars and the predicted values in the previous and the updated versions of learning data.

The separate calculation of validation fit does not change the weights of features and exemplars. Therefore, the predicted values do not change either.

Recalculation of validation fit does not change weights for features and exemplars

95. Add a text field named 'Exemplars_used' to the table of Baltic precipitation data.

Use a field size of 255 characters.

96. Restart the calculation of validation fit.

The VID number and similarity of exemplars used to calculate the predicted value for one or another observation station is recorded to the *Exemplars_used* field. The predicted amount of precipitation as a numerical variable is calculated as the weighted average of the precipitation level at the exemplar stations (Chapter 4.1.2). Similarities between stations are applied as weights.

The learning of plant cover units differed from the learning of precipitation data by using samples. The next tasks ask you to calculate validation fit for the dependent variable FID = 3 (plant cover units) and to compare the result with the learning results. Part of the predicted values should change since all learning data are included to the separate calculation of validation fit. The predicted values are calculated only within the validation sample while learning. The singles within a pre-classifier category remain unpredicted even if these cases are selected to be exemplars, because there are no other exemplars within that category and no case can be predicted by itself. Predicted values of cases within a pre-classification category may also be absent if the category involves no exemplars selected by machine learning.

No case can be predicted by itself.

97. Make a copy of the Plant_cover and LOGI_F3 tables. Add a text field entitled [Plant_cover].[Exemplars_used].

98. Switch [DEP_VAR].[calc] = True where FID = 3.

99. Start calculation of validation fit.

100. Compare the weights of exemplars and the predicted values in the initial and updated versions of plant cover data.

As a result, the predicted values should be changed and calculated for all observations. Some cases have a predicted value equal to 255, which means it was not possible to give a prediction. A common cause of unpredicted values, if a pre-classifier was included, is the absence of exemplars within a pre-classifier category.

The predicted values were altered during separate calculation of the validation fit for cases not included to the validation sample of the best learning iteration. These values remained in the field *pred* of the learning data from a previous learning iteration, which was temporally the best.

Separate exemplars for every category are necessary if pre-classifying categories are involved.

5.3. Reselection of exemplars

Reselection and re-weighting of exemplars can help in finding the best predictive set when the hitherto best estimates have been obtained using a sample of observations while learning, or when the learning data have been altered. An alternative to reselecting exemplars is to continue machine learning with the updated learning data.

101. Make one more copy of the Baltic precipitation data and the log table related to this dataset.

102. Turn on calculation of the dependent variable 'Baltic precipitation' and all other dependent variables off. Start reselection of exemplars in Constud.

103. Compare the weights of exemplars and features, the predicted values and the prediction fit before and after reselection of exemplars.

The set of exemplars has changed totally. The present algorithm of exemplar reselection in **Constud** needs a revision.

Questions

1. Does the order of records in the log table change when the records are reordered in the **Constud** log table browsing window?

2. Compare the goodness-of-fit of precipitation calculated while learning and after the data from five stations were added and the separate validation was calculated.

3. Compare the goodness-of-fit of recognition of plant cover units calculated while learning and after calculating validation fit from all learning data.

4. Are the weights of features altered when learning records are added and validation fit is recalculated?

5. Will the number of exemplars change as a result of calculating validation fit?

6. In which case will the predicted values alter as a result of calculating validation fit?

7. According to which stations was the predicted value of precipitation calculated for the meteorological station on Kihnu Island?

8. Why are some exemplars used to derive the expected plant cover unit common within the pair of observation sites VID = 130 and 136, and within the pair VID = 162 and VID = 184; while being totally different between these pairs?

9. List the names of features selected by **Constud** learning to the predictive set of each variable you studied in this chapter.

Predictions

6

108	5	1	42	2		58	160	0	77	4	1	7	33		79	6		6
5	2	44	3	##	#	55	45	##	52		##	8		##	2	5	7	6
44	2	88	###	6	68	6	##	4	##	44	##	##		##	4	55	9	
2		44	###	7	99	78	##	48	##	0	##	##		##		5		33
45	48	3	###	6		8	##		##	45	##	4	##	##	333		6	5
4	45	99		##	#	46		##		76	##	8		##	6	6	42	33
15	8	1	4	1	0	79		0	0		7	0		166	0	77		7
48		4	4	0	4	0	11	0	46	0	0	55	66	88	4	150	79	44
	19	66	###	99	#	###	###	###	#	##	5	##	5	##	##		79	
46	13	###	2	18	21	0	###	2		##	2	##	5	##		#	89	7
49		###	8	0	8	0	###	85	1	##	89	##	4	##	11	##	99	44
2	3	78	###	0,5	45	15	###	22	0	##	65	##		##	55	##		2
15		5	0	###	413		###	2		##	0	##	88	##	11	##	2	44
45	13	1		###	22		###	89	0	##	2	##	0	##		#	99	1
45		65	###	22	60	120	###	2		#	###	#	0	##	##	0		1
22	22		7	22	12	7	22		44			22		0	25		22	4
0	7	545	2	22	21	8	2	7	22	1	212		212	0	0	89		1

After you have decided to calculate predicted values and/or similarities to the **Constud** database you should ask: 1) is the predicted variable tied to locations, 2) are the predicted values the main goal, or 3) are similarity values to a given category the target? Although **Constud** does not use coordinates of observations while calculating predictions, the difference in data structure between spatial and non-spatial variables remains — the type of variable (nominal/numerical) is decided according to the pattern index in case of spatial data; but according to the data layer attribute in case of non-spatial variables. In addition, temporal limits are not applied for non-spatial data layers.

The *zero similarity* and *similarity range* parameters should have the same value in learning and for prediction of the same variable (Chapter 4.1.1 and 4.2). The default values *zero* and *two* should fit best for most tasks and data.

For the prediction task, you have to switch on the predictable variable in the field *[DEP_VAR].[calc]* — the same row in *DEP_VAR* that was switched on while learning the variable. Notice also that the exemplars, their weights and the feature values are located in the table or view whose name is given in the field *[DEP_VAR].[dataquery]*. The predictable cases are in a separate table or view, the name of which should be written in a text box in the *Knowledge Application* panel of **Constud**'s main window (Fig 6-1). The predictable cases must have: 1) fields for all features included to the best predictive set, 2) the field *VID* — for unique observation numbers, 3) field **pred** — for predicted values and 4) field **sim** — for similarity values. The field *F* which contains codes of nominal categories is necessary when similarity to a given category is to be calculated. The feature weights are read from the log table row whose ID number is in *[DEP_VAR].[Log_ID]*.

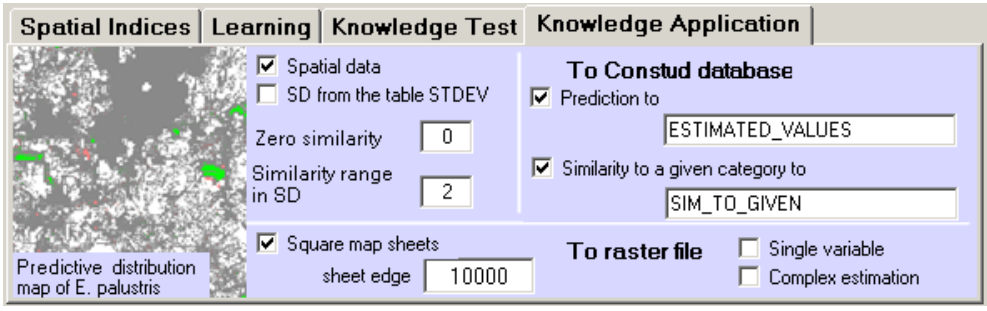


Figure 6-1. Settings for knowledge application in the **Constud** main window.

6.1. Boolean variable

Following this tutorial, you have prepared datasets of two Boolean variables: presence/absence sites of *Epipactis palustris* and enterobiasis positive or negative children. The presence/absence sites are bound to spatial coordinates and described

by location features. The second dataset is an example of non-spatial data.

6.1.1. Spatial Boolean variable

Constud produces estimates according to similarity with exemplars. A list of locations has been prepared for estimating; information regarding which cases are more similar to the known *E. palustris* presence sites and which ones to absent sites can be found in the *Predictable_sites* table in the database of tutorial data.

104. Import the Predictable_sites table to your Constud database.

There are only VID numbers, X and Y coordinates in this table. You should add fields for features and to calculate the feature values yourself this time.

105. Add the necessary fields: [pred], [sim], [from], [until] and also the fields for features to the Predictable_sites table. Switch off calculation of all variables except E. palustris in the field [DEP_VAR].[calc].

According to this tutorial, features with AID = 1–9 and AID = 11–12 were used for learning the presence/absence of *E. palustris*. Actually, only the features listed in the log table *F_weights* field in the row which has an ID number recorded to the *DEP_VAR* table are necessary for prediction, although excessive features do not harm the system. The following query adds all features that were involved in **Constud** learning since the features selected to the predictive set may be different.

```
ALTER TABLE [Predictable_sites] ADD
[pred] byte NULL, [sim] byte NULL, [From] byte NULL,
[Until] byte NULL, [1] byte NULL, [2] byte NULL,[3] byte NULL,
[4] byte NULL,[5] byte NULL,[6] byte NULL, [7] byte NULL,
[8] byte NULL,[9] byte NULL,[11] byte NULL,[12] byte NULL;

UPDATE [DEP_VAR] SET [calc]= False;

UPDATE [DEP_VAR] SET [calc]= True WHERE FID = 2;
```

106. Ensure that features with AID = 1–9 and AID = 11–12 are switched on in the field [EXPL_VAR].[F2]. Select calculation of spatial indices to the table of predictable sites (Fig 6-2).

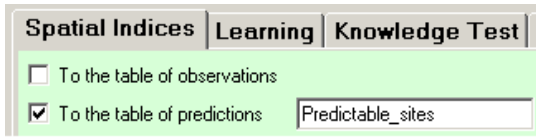


Figure 6-2. Constud settings for calculating indices to the table of predictable sites.

107. Ensure that the features received values in the Predictable_sites table and then start prediction of spatial data to the Constud database (Fig 6-3).

Number 1 in the *pred* field indicates a larger similarity of the location to the exemplar sites of *E. palustris* occurrence; zero means a greater similarity to the exemplar sites of the species' absence. The *sim* field contains the mean similarity [%] of locations to the most similar exemplars used to calculate the prediction. The number of exemplars used is determined by the sum of similarity adjusted during **Constud** learning (details in Chapter 4.1.2). The features of exemplars are in the database object whose name is in *[DEP_VAR].[dataquery]*. Only the features together with the weights recorded in the *F_weights* field in the log table in the row with the best learning result are used in prediction.

Similarity to exemplars represents decision certainty,
not similarity to find sites.

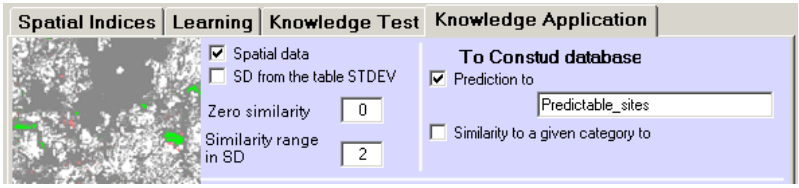


Figure 6-3. Constud settings for predicting a spatial variable to the database object *Predictable_sites*.

6.1.2. Non-spatial variable

In Chapter 4.8, **Constud** learned to recognise enterobius positive and negative children as an example of non-spatial data. Explanatory features of three putative children are prepared for a prediction exercise in Table 1. **Constud** requires a database object with fields listed in the beginning of Chapter 6 for predictable cases. A simple option for creating the necessary table is to copy the structure (without data) of the table wherein the learning data of the same variable were mounted. The *F*, *w* and *actuality* fields are not used in prediction — you can remove these fields.

108. Copy the structure of the Enterobiasis table to a new table called 'Three_children'. Of course, you can give the table any other name. Delete fields F, w, and actuality from the new table.

109. Insert the data from Table 1 to the Three_children table in the Constud database.

110. Switch on the dependent variable 'Enterobiasis' in the field [DEP_VAR].[calc]. Switch off all other variables.

Table 1. Data for similarity-based estimating of enterobiasis positive/negative cases among nursery children.

AID	60	61	62	63	64	65	66
VID	Age	Gender	Number of children in family	One-age group	Age interval in the group	Region	Settlement type
1100	2	0	1	yes	1	1	1
1101	4	1	3	no	3	3	3
1102	6	1	5	no	4	5	5

Gender: 0 — female, 1 — male; region: 1 — Tartu, 3 — Tartu rural, 5 — Põlva rural.

111. Open Constud. Select 'Knowledge Application'. Switch off spatial data. Input the name of the table where the predictable cases are located (Three_children).

112. Start calculation and after that check the predicted values at [Three_children].[pred].

A predicted value of *True* means the child is more similar, regarding the weighted features in the predictive set, to the contaminated exemplars. *False* indicates a higher similarity to the exemplars of healthy children. Similarity percentage in the *sim* field indicates the mean similarity to the exemplars used for prediction.

6.2. Multinomial variable

The prediction of a multinomial and continuous variable is principally the same as it was in the previous examples. A predictive set of feature weights and exemplar weights must exist in the **Constud** database. The predictable variable must be switched on in the *DEP_VAR* table, and then you have to launch **Constud** and input the name of the object containing predictable cases.

Let's try it with plant cover data. In Chapter 4.5, we limited the learning cases to include only records where VID < 301. There are 500 records in the *Plant_cover* table and the rest of the records are available for a validation kit to compare the predicted and the observed values in an independent sample.

113. Switch on the dependent variable 'Plant_cover' (FID = 3) in the field [DEP_VAR].[calc]. Switch off all other variables.

114. Create a database view selecting plant cover records where VID > 300. Save the query and give it a name, e.g. Plant_cover_test.

```
SELECT * FROM [Plant_cover] WHERE VID>300;
```

115. Open Constud. Select the panel: Knowledge Application. Switch on Spatial data. Input the name of the table where the predictable cases are (Plant_cover_test).

116. Start calculation and then compare how often the predicted and observed values match.

There are quite few matches. Decision mismatches are common even among specialists visiting the same sites in a diverse natural or semi-natural landscape if a hundred or more detailed plant cover categories are given to choose from. There is no prospect for recognizing so many different units using only the remote sensing and cartographical data offered as example data for this tutorial.

6.3. Numerical variable

The prediction of a numerical variable differs from other types of dependent variables in **Constud** by the format of the field *pred* — single not byte, and by the value stored when the prediction cannot be given — the mean value of the dependent variable in learning data instead of 255.

There is a *Baltic_precip_test* table in the database of tutorial data containing the mean annual amount of precipitation in millimetres, features describing location and vicinity of the meteorological stations, and fields necessary to store predictions.

117. Switch on the dependent variable 'Baltic precipitation' (FID = 4) in the field [DEP_VAR].[calc]. Switch off all other variables.

118. Import or copy the Baltic_precip_test table from the database of tutorial data to the Constud database.

119. Open Constud. Select 'Knowledge Application'. Switch on spatial data. Input the name of the table where the predictable cases are located (Baltic_precip_test).

120. Start calculation and then look through the predicted values. Are the predicted values mostly higher at stations where the observed values are higher or vice versa?

6.4. Single category of a multinomial variable

Following task 69 of this tutorial, a query was created to view the first 300 records from the *Plant_cover* table and to rename the fields: *pred1*, *sim1*, *w1* and *actuality1*.

121. Copy the database view 'Plant_cover_learning1', name it 'Plant_cover_test1', and modify the copy to select records where VID > 300.

Fields *w* and *actuality* can be omitted since they are not used for prediction. The *F* field is useful for comparing the observed and predicted values. **Constud** does not use (does not crib from) this field while calculating predictions.

The query should be:

```
SELECT [VID],[F],[X],[Y],[pred1] AS pred, [sim1] AS sim, [From],
[Until],[13],[14],[15],[16],[17],[18],[19],[20],[21],[22],[23],[24],
[25],[26],[27],[28],[29],[30],[31],[32],[33],[34],[35],[36],[37],
[38],[39],[40],[41],[42],[43],[44],[45],[46],[47],[48],[49],[50]
FROM Plant_cover WHERE [VID]>300;
```

Notice that the view renames fields *pred1* and *sim1*, and passes on records where VID > 300. The first 300 records were used for learning.

122. Switch on all rows of the dependent variable 'Baltic precipitation' (FID = 5) in the field [DEP_VAR].[calc]. Switch off all other variables. Check whether feature 13 is recorded as the preclassifier where [DEP_VAR].[FID]= 5.

123. Open Constud. Select 'Knowledge Application'. Switch on spatial data. Input the name of the table where the predictable cases are located (Plant_cover_test1).

124. Start the calculation and then look through the results.

Notice that rows with certain values in the field *[13]* (basic map areal category) do not have predicted values. The number 255 means the value is undetermined. There are no exemplars for these areal categories in the predictive set.

The predicted codes represent only the categories selected in the *DEP_VAR* table. This time we had only five categories in learning and in prediction. All records that had any similarity to one of these were classified to the most similar category. If only one category was selected in *DEP_VAR*, then all cases with any similarity were assigned to this category. Therefore, the separate learning of categories is more reasonable when the more common categories are included both to learning and to prediction.

6.5. Similarity to a given category

Only the rows having one and the same FID number can be switched on when calculating the similarity to a given category. **Constud** will send a warning if more is selected or no row is selected. Similarity to only those categories which have passed **Constud** learning as separate units and which are switched on in *[DEP_VAR].[calc]* can be calculated. Every category has its own predictive set of feature weights and exemplar weights, the list of features may overlap, and no exemplar of separate categories is the same.

In addition to the fields that are necessary for prediction, field *F* containing the code of a given category is obligatory to calculate the similarity to a given category (Fig 6-4). The data structure prepared for the *similarity to a given category* option differs from the data structure needed for **Constud** prediction only by the necessary field *F* and by the unnecessary field *pred*.

Constud looks for the same FID, the same learning data, the same pre-classifier, if applied, and the same log table for all categories. Similarity to a given category will be calculated only for those cases in which the given category is switched on in *DEP_VAR* and in which the pre-classifying unit contains exemplars of this given category.

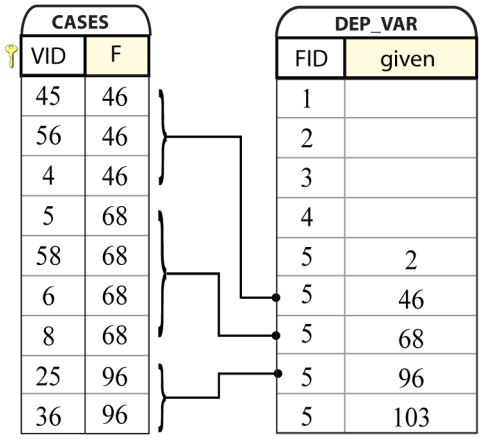


Figure 6-4. Linkage between *[DEP_VAR].[given]* and the field for given code *F* in the database object containing cases for similarity estimation.

125. From the field [DEP_VAR].[calc], switch on the row where FID = 5 and the land cover unit is fallow. Switch off all other rows.

126. Clear the field [Plant_cover_test1].[sim].

```
UPDATE Plant_cover_test1 SET [sim] = Null;
```

127. Open Constud. Select Knowledge Application. Switch on spatial data. Input the name of the database object where the predictable cases are located (Plant_cover_test1) (Fig 6-5). Start calculation. Leave Constud open. Refresh the list of predicted values.

Similarity is zero in some categories of the pre-classifying feature AID = 13 because of missing exemplars in these categories.

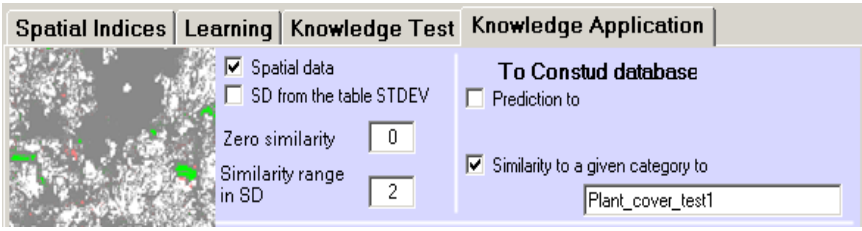


Figure 6-5. Constud settings for estimating similarity to a given category of a spatial variable to the database object *Plant_cover_test1*.

128. Order 'Plant_cover_test1' according to values in F and move to records of the plant cover unit fallow (F = 96) or select fallow fields from Plant_cover_test1 using the following query.

```
SELECT [F], [pred],[sim],[13] FROM [Plant_cover_test1] WHERE [F]=96;
```

129. Remove the pre-classifier from all rows where FID = 5.

Constud's present version does not accept different pre-classifiers in separately handled categories of a nominal variable.

```
UPDATE [DEP_VAR] SET [precl]=Null WHERE FID=5;
```

130. Repeat the calculation of similarity. Leave Constud open. Refresh the list of predicted values.

The similarity values should be above zero for all instances of fallow, regardless of feature 13.

131. Switch on all rows in DEP_VAR where FID = 5.

```
UPDATE [DEP_VAR] SET [calc]=True WHERE FID=5;
```

132. Restart the calculation of similarity. Leave Constud open. Refresh the list of predicted values.

Similarity is calculated for all records which have a code in the *F* field matching a value in *[DEP_VAR].[given]*.

133. Update the field [Plant_cover_test1].[F] to a common value, e.g. 96 (fallow).

```
UPDATE [Plant_cover_test1] SET [F]=96;
```

134. Restart the calculation once more. Refresh the list of predicted values.

Similarity of every location among the predictable cases has a calculated similarity to the exemplar sites of fallow, regarding the features adjusted to recognise fallow, and not using pre-classes.

Questions

1. What is the range of values for the similarity to exemplars of a Boolean variable?

2. Which of the children listed in Table 1 are probably enterobiasis positive, and which negative?

3. Which features were used by **Constud** to recognise enterobiasis positive/negative cases in task 111?

4. What is the relative extent of overlap in percentages between observed and estimated plant cover categories in task 116?

5. How large is the mean similarity between exemplars and cases in task 116: 1) where does the estimated plant cover category match the observed one, 2) where it does it not match? Omit the missing data value — 255.

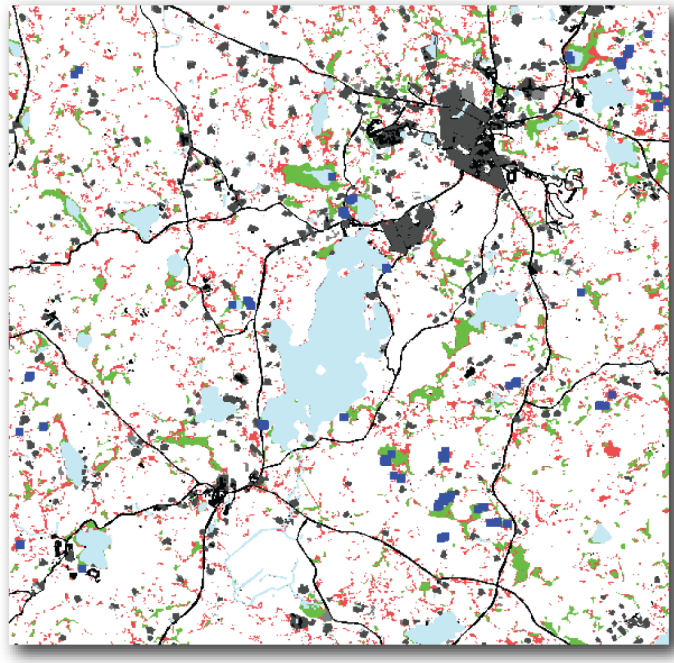
6. How large in millimetres is the difference between similarity-based estimations and observed levels of the mean precipitation according to the results of task 120?

7. Which are more variable — similarity-based estimations of a numerical variable or the observed values? The elementary statistics to measure variability are range and standard deviation.

8. Select observations wherein fallow has been recorded as the plant cover unit. Within which pre-classifier category (feature 13) are these observations sites most similar on average to the exemplars of the category fallow? The codes of the Estonian basic map categories are listed in the *Base_map_areas* table in *Constud_tutorial_data.mdb*.

9. In which sense are the predicted cases used in Chapter 6.2 forming an independent sample?

Maps



For creating predictive maps in **Constud** you need: 1) a **Constud** database containing weights for features and exemplars, 2) dependent variable(s) selected in the field *[DEP_VAR].[calc]*, 3) data layers of features and of a pre-classifier, if the last is included into the predictive set as rst-format raster files. In addition, a layer of interpolation polygons can be involved (Chapter 7.1). The database containing weights is normally the result of **Constud** learning.

The same variables learned in previous tasks of this tutorial are used in the following exercises. We suggest using the same database. You should also be familiar with how to select the dependent variable in *[DEP_VAR].[calc]*.

For mapping a dependent variable using a predictive system, one needs the values of explanatory variables for every single location of the map area. **Constud** can calculate the feature values for every location (pixel of the output raster) according to the feature attributes in the *EXPL_VAR* table using pixel values in data layers. The data layers can be located on an FTP server or in a local folder. The prefix and the data source (FTP or local) defined in **Constud**'s main window are applied. In the map settings window, you must input full names of the files and these files must be accessible through a local network.

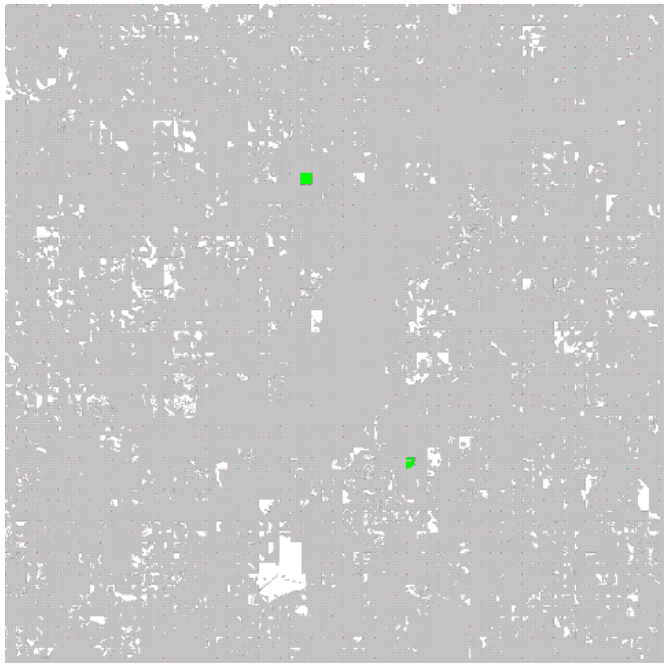
7.1. Interpolation

The time spent for forming a predictive map depends on the number of pixels in the output raster since the prediction should be calculated at every output pixel. Usually the number of pixels is large and the variability of output area is different. Some regions are relatively diverse; others are much more or totally uniform. A dense grid of calculated locations is needed only at diverse regions but a larger distance between calculated locations would be enough for more uniform regions.

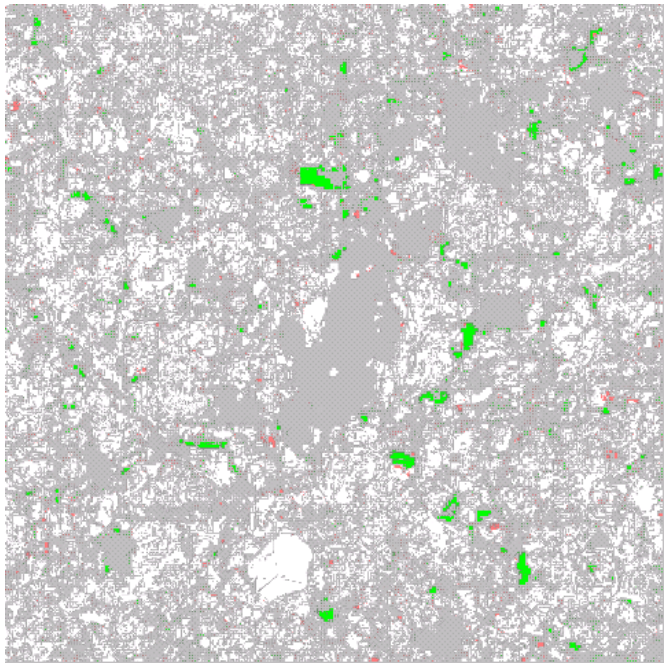
Constud offers an algorithm which controls the density of directly predicted pixels. The **Constud** user can select the initial density of calculated pixels by setting the initial (the largest) pixel interval. The interval is equally valid in rows as in columns. If the initial interval is, for example, 8, then the predicted value is calculated first at the crossing point of every eighth row and eighth column. After that, **Constud** checks whether the predicted category of a nominal variable is the same at all four corners, or whether the difference in predicted values of a numerical variable is below a given level. **Constud** also checks the similarity of the corners to the exemplars used for predicting. If interpolation polygons are used then all corners must also belong to the same category in the interpolation layer. If the four corners obey to these restrictions then **Constud** omits predictions for the intermediate output pixels. In case of a numerical variable, the predicted values calculated at corners will be interpolated; in case of a nominal variable all intermediate pixels are assigned to the same category. In regions not fitting the restrictions allowing interpolation,

the similarity-based prediction is repeated at a spatial interval twice shorter. The process is repeated until all pixels have received a predicted value. The interpolation algorithm is described also in Chapter 5.6.3.1 in *Remm et al. (2010)*.

A)



B)



C)

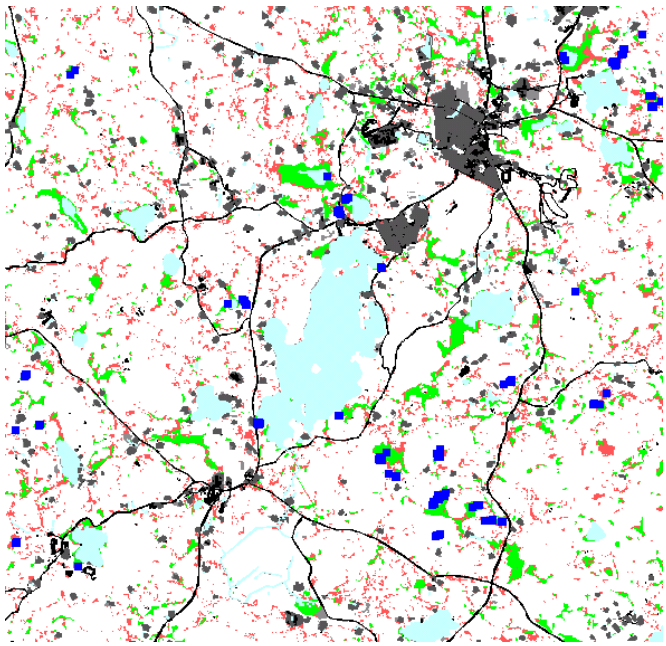


Figure 7-1. Forming a predictive map depicting similarity to the find locations and to the absence locations of *Epipactis palustris*. **A** — interpolation interval 16 pixels, **B** — interval 4 pixels, **C** — completed map including additional layers. Light-grey — uncalculated area, white — more similar to absent locations, green — more similar to find locations, pink — equally similar to find locations and absent locations, light-blue — water bodies, dark-blue — find locations, charcoal— settlements, black — major roads. Predictions for water bodies and built-up areas were impeded by the pre-classification layer. Modified from *Remm et al. (2010)*.

The file containing interpolation polygons must have the same number of pixels, the same pixel size and the same border coordinates as the output rasters. Rasterised areas from the Estonian basic map sheet 5454 are used both as a pre-classifier and as the layer of interpolation polygons in the next tasks. You should download the file because for **Constud** the file containing interpolation polygons must be in the user's computer or accessible through LAN.

135. Get files 5454_40m.rst and 5454_40m.rdc from the learning data available at the Constud web site. Make note of the folder where you saved these files.

7.2. Single variable

There are two options in the *Knowledge Application* tab panel of **Constud**'s main

window: *single variable* and *complex variable* (Fig 7-2). A predictive map and a similarity map is output for each selected variable. For predictive mapping of a complex variable, **Constud** will look for separate categories of the same complex variable selected in *[DEP_VAR].[calc]*. In addition to a predictive map, **Constud** always calculates a similarity map indicating the mean similarity between a location and the exemplars used for calculating the predicted value.

7.2.1. Boolean variable

Two different Boolean variables were used in the previous tasks of this tutorial — enterobiasis positive and negative children and presence/absence of *Epipactis palustris*. The enterobiasis observations are not related to spatial coordinates and cannot be mapped in **Constud**.

136. Switch on the field [DEP_VAR].[calc] in the row where the dependent variable is Epipactis palustris (FID = 2) and off in all other rows.

137. Start Constud; select the database and single variable in the Knowledge Application tab panel of Constud's main window (Fig 7-2). Press Continue.

The window containing settings for map generation opens (Fig 7-3). The name of dependent variable is at the upper edge of the window. The user can set the following attributes.

Map sheet — map sheet number. **Constud** looks for rst-files with a name which equals the map sheet number from the folders of data layers. The folder names are in the *D_LAYERS table*; the folder prefix is added (Chapter 1-2).

Pixel edge — the length of pixel edge in the same units as the coordinates of map sheet boundaries and observations.

Difference in similarity — maximum accepted difference [%]. Pixel values are not interpolated if the difference in similarity values between corner pixels is larger.

Initial distance of interpolation — expressed in pixels. The distance is divided by two in next iteration.

Rst-file of estimated values — the name of the **Idrisi** raster file where the predicted pixel values will be stored.

Rst-file of similarity to exemplars — the name of the **Idrisi** raster file where the similarity of every location to the exemplars used in prediction will be stored.

Interpolation polygons — controls visibility of the textbox for input.

Rst-file of interpolation polygons — the name for the file containing categories used to delimit interpolation (Chapter 7.1). The file of interpolation polygons must have the same boundaries as data layers and the same pixel size as the output rasters.

Do not calculate where pre-class = — input an exclusion value in the pre-classifier layer. The exclusion value in Fig 7-3 is 48, marking lakes. You can try a different exclusion value; for example, the code 59 denotes ponds, 108 — fields, and 254 — industrial yards. Unfortunately, **Constud** does not enable concurrent use of more than one exclusion code per data layer. An additional option for exclusion is described in Chapter 7.2.2. Categories have to be merged to exclude more than one category in a pre-class or interpolation layer. One of these layers can contain only two values marking area to calculate and area not to calculate, thus delimiting the calculated area of the output map.

Bytes per pixel — the number of bytes per pixel in the file of interpolation polygons. The value one denotes one-byte format (values 0–255), and value 2 short integer (values −32,768…+32,767).

Save interim layers — if switched on then the results from every interpolation interval are stored in a separate file. The interpolation interval is added to the name of the output file.

Warn if a data layer is missing — an option to ask for a message every time **Constud** did not find a necessary data layer. Features from the missing data layers are omitted from similarity calculations; decisions are made using the existing features.

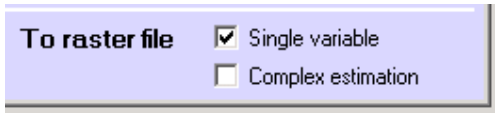


Figure 7-2. Settings for predictive mapping of a single variable in **Constud**'s main window.

138. Set the settings for calculating presence/absence map of Epipactis palustris as shown in Fig 7-3. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Press Calculate to start calculations.

This calculation takes a few minutes. The **Constud** main window reopens when the calculation is completed. If the field *[DEP_VAR].[calc]* contained more than one selected row, then the window of settings for map calculations would open offering to start map calculation for the next variabThe time needed for a map calculation depends on processor speed, the number of output pixels to calculate, the number of data layers, the number and the computational complexity of features, and also on the connection speed if the data layers are on an FTP server.

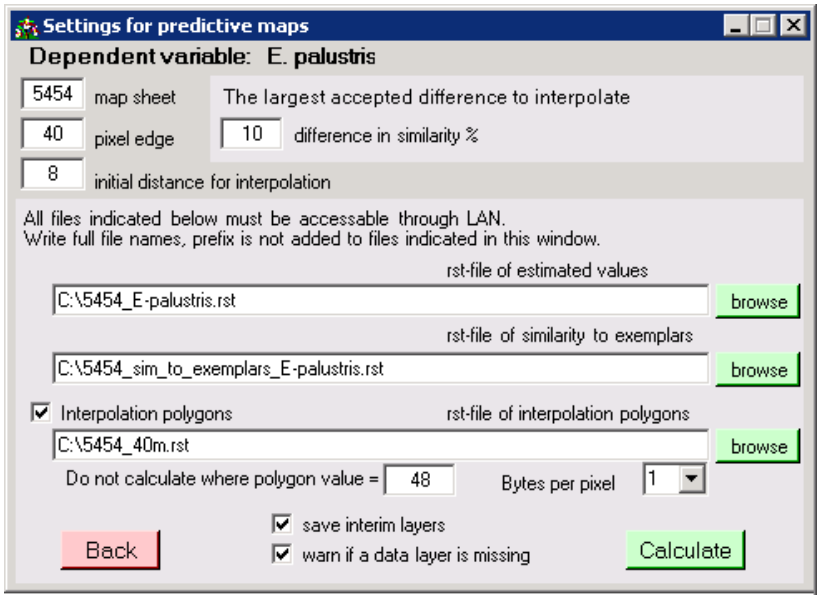


Figure 7-3. Settings for calculating a predictive map of *Epipactis palustris* presence/absence for map sheet 5454.

139. To view and convert Idrisi rst-files produced by Constud the matching rdc-files are required.

Here you have at least two options: to copy and rename the existing file *5454_40m.rdc* (see task 135) or to create the rdc-file from scratch. The second option can be implemented by copying the following text to a simple text file (**Notepad** format). Give the file the same name as has the corresponding rst-file, and change the file extension from txt to rdc.

file format.: IDRISI Raster A.1
file title..:
data type...: byte
file type...: binary
columns.....: 250
rows.....: 250
ref. system.: plane
ref. units..: m
unit dist...: 1.0000000
min. X.....: 640000.0000000
max. X.....: 650000.0000000
min. Y.....: 6450000.0000000
max. Y.....: 6460000.0000000
pos'n error.: unknown
resolution..: 40.0000000

min. value..: 0
max. value..: 255
display min.: 0
display max.: 255

140. Have a look at the LOG-file to see which features were applied. Open the predictive map and the similarity map and think about why the output maps are the way they are.

You can see the interim results in the files which have a name ending with the number of pixels in the interpolation interval.

Notice that the similarity map of a Boolean variable represents similarity to the predicted value and not similarity to the find sites of the species. The similarity in the map is similarity to the exemplars of find sites in locations where the predicted category is species presence, and similarity to the exemplars of absent sites in locations where the predicted category is species absence.

7.2.2. Multinomial variable

In Chapter 4.5, **Constud** learned plant cover units as the example of a multinomial variable. The Estonian basic map areal categories were involved as a pre-classifier. **Constud** reads the pre-classifier from the *DEP_VAR* table and turns the textbox for pre-classifier input to visible if a pre-classifier is recorded in the field *[DEP_VAR]. [precl].* In addition to separate exemplars for each category, the pre-classifier, as well as the layer of interpolation polygons, enables users to exclude a part of the rectangular area in a raster file from calculations.

141. Switch on the row in [DEP_VAR].[calc] where FID = 3 and all other rows off.

142. Start Constud, select 'Single variable' to 'Raster file' from the Knowledge Application tab panel of the Constud main window (Fig 7-2). Press Continue.

143. Arrange the settings for calculating the predictive plant cover map as given in Fig 7-4. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Press Calculate to start calculations.

The polygons for interpolation and the pre-classifier are the same this time, which may not always be the case. Lakes are excluded according to the layer of interpolation polygons, and fields by the pre-classifier.

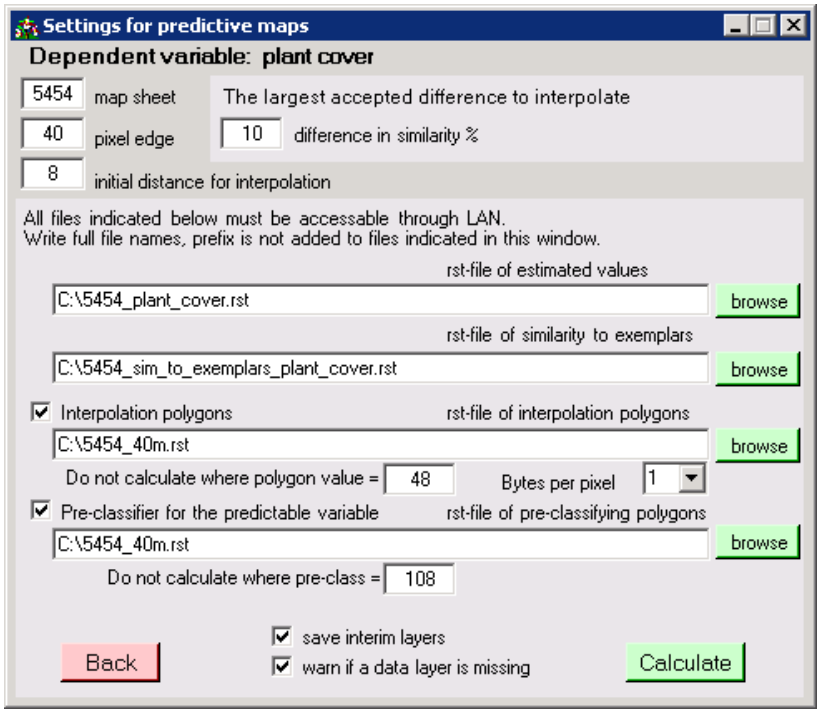


Figure 7-4. Settings for mapping predictive plant cover in map sheet 5454.

144. Add rdc-files matching the output rst-files; see task 139 for details.

145. Have a look at the LOG-file to see which features were applied. Open the predictive map and the similarity map and think about why the output maps are the way they are.

You can see the interim results in the files which have a name ending with the number of pixels in the interpolation interval.

7.2.3. Numerical variable

The maximum accepted difference in the predicted values for interpolation is among the settings when mapping a numerical variable. For a nominal variable, interpolation is restricted to the same predicted category.

The features used in learning Baltic precipitation data (task 62) were connected to universal imaginary data layers that actually do not exist. That was sufficient for **Constud** learning and for predicting to an output table, but not for predictive mapping because the values of descriptive features are necessary at every location of the output map. Feature values cannot be calculated from imaginary data layers. Therefore, before map production, we have to append the real data layers

corresponding to the features applied for Baltic precipitation mapping. After that, the layer numbers of these features in the *EXPL_VAR* table must also be corrected.

Until now, only the map sheets of the 1: 20,000 Estonian basic map are referenced in the *Map_sheets* table. The data layers used for mapping precipitation all over the Baltic countries must cover this entire area. The following exercises demonstrate how to use data layers that do not follow the Estonian basic map nomenclature and projection.

146. Copy the attributes of additional data layers 34–40 from the table Additional_layers in Constud_tutorial_data.mdb to the D_LAYERS table in the Constud database.

It is essential to follow the KID numbers, pixel size and folders of the data layers. The folder names indicate the location of data layers in ADSD. Users outside the University of Tartu should mount their own data layers.

147. Add map sheet number 1 to the Map_sheets table. The boundary coordinates of this sheet must be min-x = 304000, min-y = 5968000, max-x = 767000, and max-y = 6620000.

Constud uses only rectangular coordinates and does not deal with coordinate systems. Map sheet numbers are not connected to any map sheet nomenclature. Only matching numbering used for naming files in data layers and in the Constud database is essential.

148. Open tables EXPL_VAR and D_LAYERS. Alter the numbers of data layers in the field [EXPL_VAR].[KID] where AID is 51–57 in rows where the name of a feature matches the folder name of the data layer.

Do not alter folder names. If you have followed this tutorial, the new data layers should be in the same order as features.

149. Download files Baltimaa_koodiga_0.rst and Baltimaa_koodiga_0.rdc from the Constud web site. Remember the folder where you saved these files.

The file *Baltimaa_koodiga_0.rst* contains a mask of the territory of the Baltic countries which is coded as 0; all other area has pixel values of 255. The rdc-file contains meta-data for the rst-file.

150. Switch on the row where FID = 4 in [DEP_VAR].[calc] (Baltic precipitation) and all other rows off.

151. Start Constud and select raster map of a single variable from the Knowledge Application panel of the Constud main window (Fig 7-2).

152. Check off 'Square map sheets'. Press Continue.

153. Set settings for mapping annual precipitation in the Baltic countries as given in Figure 7-5. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 1000 m because pixels in the file of interpolation polygons have this size. Press Calculate to start calculations.

We do not use a pre-classifier this time because it was not involved while learning this dependent variable. The mask of the territory of the Baltic countries in *Baltimaa_koodiga_0.rst* (obtained at task 149) is used as the layer of interpolation polygons. The area outside the Baltic countries, but within the rectangular map, is coded with pixel values of 255. You can check off saving interim results if you do not like to spam the output folder with temporary files.

154. To view and convert Idrisi rst-files produced by Constud you must add matching rdc-files.

You can simply copy and rename the file *Baltimaa_koodiga_0.rdc* in order to have an rdc-file matching the output similarity file. For the file of predicted precipitation values, the copied rdc-file has to be changed a little because the predicted values of a numerical variable are stored as single-precision real numbers using four bytes per pixel. Make the following replacement in the rdc-file of predicted values: „*data typ...: byte*“ => „*data type...: real*“.

155. Have a look at the LOG-file to see which features were applied. Open the predictive map and the similarity map and think about why the output maps are the way they are.

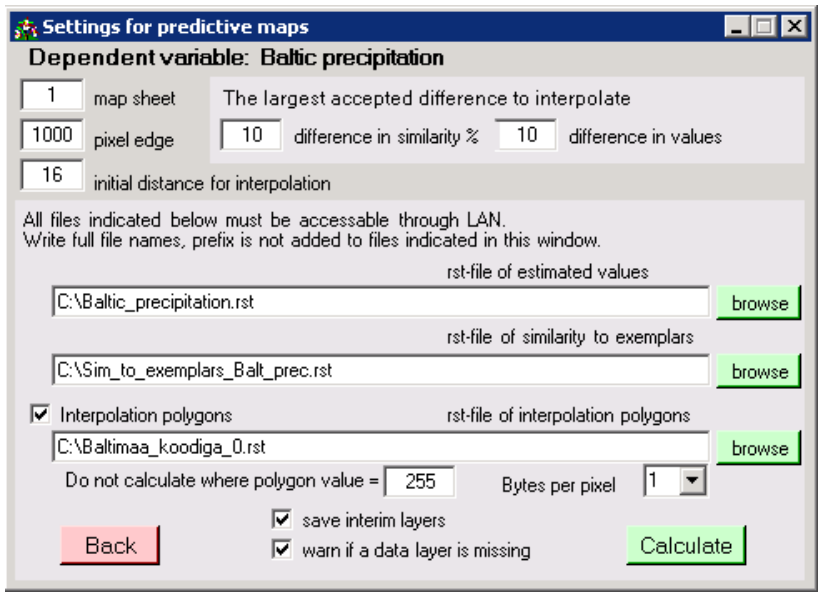


Figure 7-5. Settings for mapping Baltic precipitation.

7.2.4. Single category of a nominal variable

Constud treats a single category of a nominal variable as if it were a Boolean variable, if the mapping process is started as a prediction of a single variable (not a complex variable) (Fig 7-2). It means that a separate map is calculated for every category that completed separate learning in **Constud**.

156. Switch on the row in [DEP_VAR].[calc] where the separately learned plant cover units are boreo-nemoral grassland, fallow and field. Switch all other rows off. Restore preclass 13 where [FID] = 5. This field was cleared following task 129.

157. Start Constud, select 'Single variable' to 'Raster file' from the Knowledge Application tab panel of the Constud main window (Fig 7-2). Ensure that the option 'Square map sheets' is switched on. Press Continue.

158. Arrange settings for map calculation as in Fig 7-6. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Press Calculate to start calculations.

Notice that the name of the dependent variable is the first selected plant cover category selected in the *DEP_VAR table* — the boreo-nemoral grassland this time. The same pre-classifier as in learning is also the right one to involve in map production. If no pre-classifier was involved in learning, then there is little reason to use a pre-classifier in mapping. You can exclude interpolation polygons and saving interim results. The map calculation passes lakes (code 48) and industrial

yards (code 254) according to the settings in Fig 7-6. You can choose different areal categories for exclusion as void. The Estonian basic map's areal category codes are listed in the *Areal_categories* table in the database, *Constud_tutorial_data.mdb*.

The settings window for map making reopens, displaying the name of the next selected category when the maps of the current category are completed.

159. Modify the names of output files to match the next plant cover category and restart map calculation.

160. Add rdc-files to the rst-files of final results.

The files of final results do not have the number of interpolation interval at the end of the file name. You can copy any existing rdc-file matching map sheet 5454, **Idrisi** byte format raster, and a 40 m pixel edge.

161. Compare the output files from tasks 158 and 159.

According to this tutorial, the name of output files corresponding to the first category should be: *5454_grassland.rst* and *5454_sim_to_grassland_exemplars.rst*.

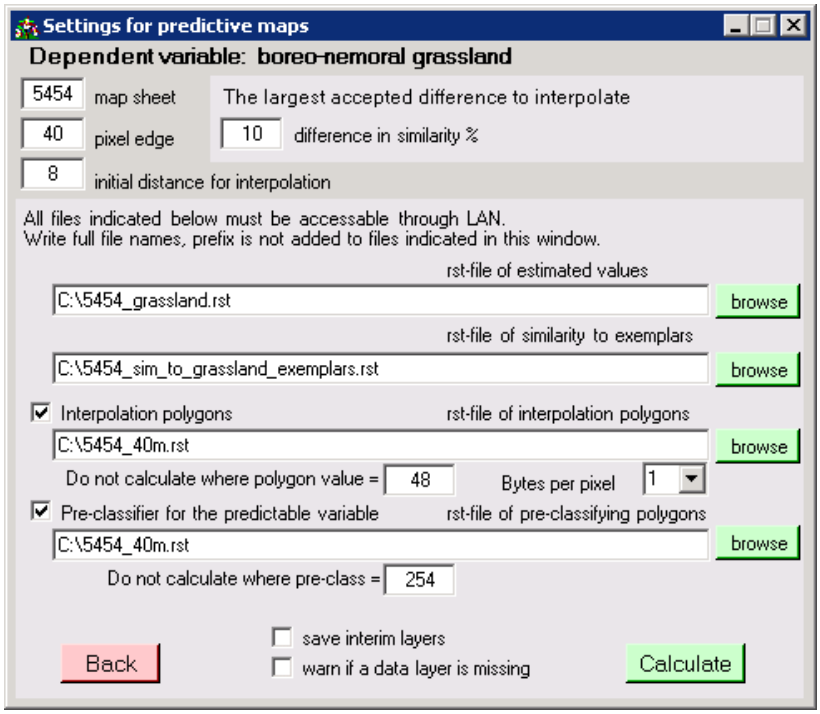


Figure 7-6. Settings for mapping the distribution of boreo-nemoral grasslands.

A nominal variable consisting of categories that have gone through **Constud** learning

separately is treated as a complex variable in the next chapter, not as a set of Boolean variables — one per each category, as was the case in this part.

7.3. Complex estimation

The calculation of predictive maps for a nominal variable for which categories have gone through **Constud** learning separately (complex variable) differs computationally from predicting a single variable. Therefore, it is a separate option in the **Constud** main window (Fig 7-7). The option *Complex estimation* can be used only while the dependent variable has *ftype* = 4 in the field *[DEP_VAR].[ftype]*. After pressing *Continue* the window containing settings for predictive mapping opens. This window has three slightly different forms (Fig 7-8).

The first form (Fig7-8A) represents settings for mapping the most similar unit among categories selected in *[DEP_VAR].[calc]*. A map depicting similarity between locations and the exemplars used at every location is an incidental output. The second form is for mapping similarity between a location and a category predetermined for that location according to an existing map (Fig 7-8B). The third option combines the mapping of predicted values and similarity to a given category (Fig 7-8C).

The predicted category of a complex nominal variable cannot be inferred at once because a different set of feature weights is mounted as the result of **Constud** learning for recognition of every category of the complex variable. At every output pixel, **Constud** calculates similarity to each selected category separately. The most similar category will be the predicted one. Complex estimation can be applied only for nominal variables with categories that have gone through **Constud** learning separately.

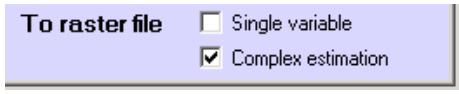
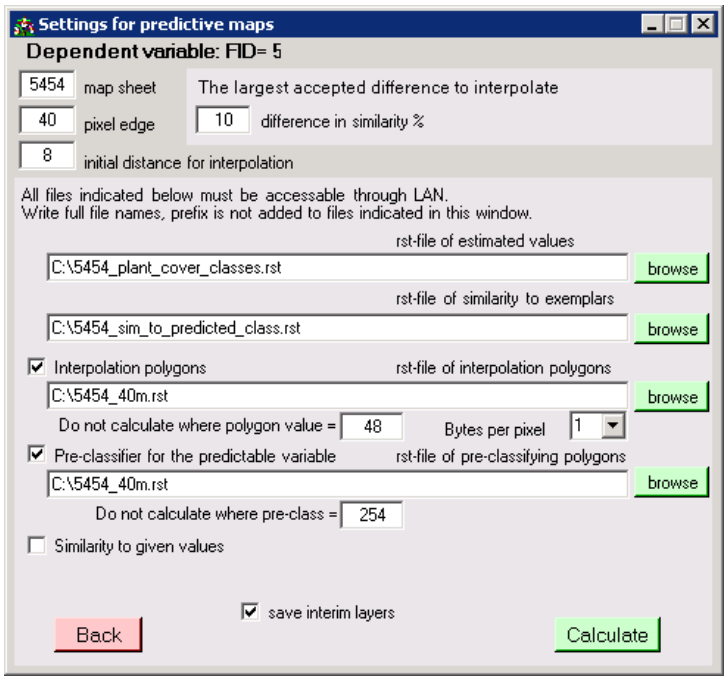
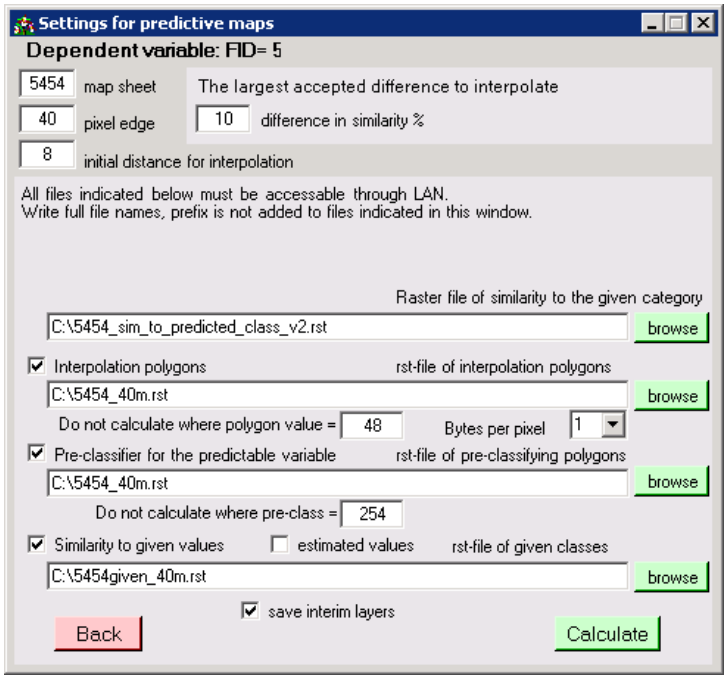


Figure 7-7. Checkbox for complex estimation in the *Knowledge Application* panel of the **Constud** main window.

A)


Settings for predictive maps
Dependent variable: FID= 5
5454 map sheet
40 pixel edge
8 initial distance for interpolation
The largest accepted difference to interpolate
10 difference in similarity %
All files indicated below must be accessable through LAN.
Write full file names, prefix is not added to files indicated in this window.
rst-file of estimated values
C:\5454_plant_cover_classes.rst
browse
rst-file of similarity to exemplars
C:\5454_sim_to_predicted_class.rst
browse
Interpolation polygons
rst-file of interpolation polygons
C:\5454_40m.rst
browse
Do not calculate where polygon value = 48
Bytes per pixel 1
Pre-classifier for the predictable variable
rst-file of pre-classifying polygons
C:\5454_40m.rst
browse
Do not calculate where pre-class = 254
Similarity to given values
save interim layers
Back
Calculate

B)


Settings for predictive maps
Dependent variable: FID= 5
5454 map sheet
40 pixel edge
8 initial distance for interpolation
The largest accepted difference to interpolate
10 difference in similarity %
All files indicated below must be accessable through LAN.
Write full file names, prefix is not added to files indicated in this window.
Raster file of similarity to the given category
C:\5454_sim_to_predicted_class_v2.rst
browse
Interpolation polygons
rst-file of interpolation polygons
C:\5454_40m.rst
browse
Do not calculate where polygon value = 48
Bytes per pixel 1
Pre-classifier for the predictable variable
rst-file of pre-classifying polygons
C:\5454_40m.rst
browse
Do not calculate where pre-class = 254
Similarity to given values
estimated values
rst-file of given classes
C:\5454given_40m.rst
browse
save interim layers
Back
Calculate

C)

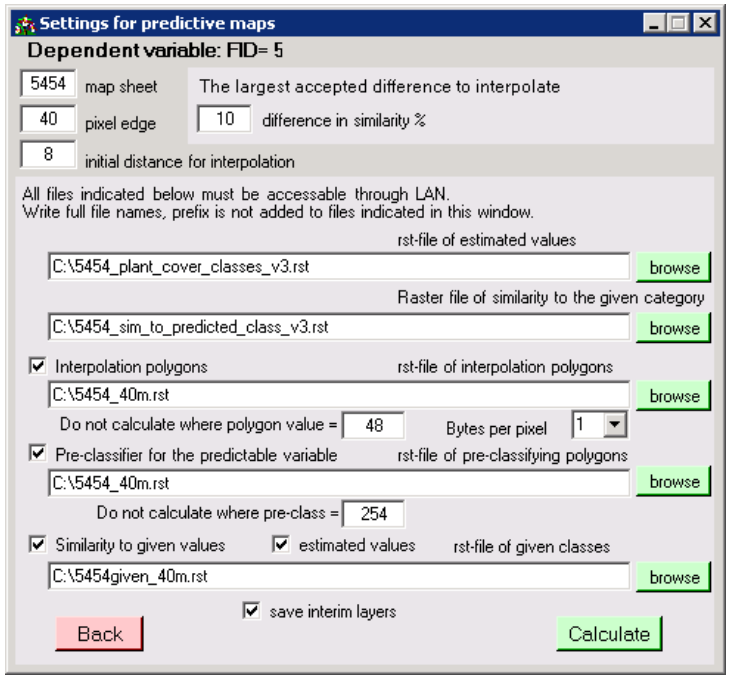


Figure 7-8. Settings for mapping complex nominal variables. **A** — predicted category and similarity between the location and exemplars of the predicted category, **B** — similarity to a category given for every location, **C** — similarity to a category given for every location and the predicted category.

162. Switch on all rows in [DEP_VAR].[calc] containing meta-data from the separately learned plant cover units (FID = 5). Switch all other rows off.

163. Start Constud and select 'Complex estimation' from the Knowledge Application panel of the Constud main window (Fig 7-7). Switch on 'Square map sheets'. Press Continue.

164. Arrange settings for map production as given in Fig 7-8A. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Press Calculate to start calculations.

Unlike previous mappings, the name of the dependent variable is missing from the map settings window because there are only names of single categories in the *DEP_VAR* table in the **Constud** database.

You can switch off the application of interpolation polygons and saving interim layers. It is correct to use the same pre-classifier as was used while learning the variable. The output is not calculated at lakes (code 48) and in industrial yards (code

254) according to the settings in Fig 7-8A. You can choose different areal categories to void. The codes of areal categories in the Estonian basic map are listed in the table *Areal_categories* in *Constud_tutorial_data.mdb*. When the maps are completed, the **Constud** main window reopens.

The next task is to make a map depicting similarity to a category given for every location as pixel values in a raster map. The vegetation map produced in task 143 or the *5454_given_40m.rst* file downloadable from the **Constud** web site are available as maps depicting examples of given categories.

165. Acquire the 5454_given_40m.rst file from the example data at Constud's web site.

This file contains a simplified theoretical plant cover model where all fields are classified as fallow, all forests as boreal pine woods and all grasslands as boreo-nemoral grasslands. All other areal categories according to the 1: 10,000 basic map have a pixel value of zero. The smaller pixel values in the calculated similarity map should indicate areas less similar to the given unit; similarity values close to 100% refer to areas probably really belonging to the given category according to the features used. The next tasks are for mapping similarity to a given category. The tasks differ only by adding the map of predicted values to the similarity map.

166. Arrange settings for map production as given in Fig 7-8B. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Saving of interim results can also be excluded. Press Calculate to start calculations.

The result is only the map of similarity to the category given in the existing map. The calculation is relatively fast.

167. Arrange settings for map production as given in Fig 7-8C. Adjust the folder names to the actual folders in your computer. Use a pixel edge of 40 m because pixels in the file of interpolation polygons have this size. Saving of interim results can also be excluded. Press Calculate to start calculations.

Similarity to a given category, as well as the predicted category, is the result. The calculation of the predicted value is relatively time consuming.

168. Add rdc-files to the rst-files output from tasks 164, 166, 167.

You can copy any existing rdc-file matching map sheet 5454, **Idrisi** byte format raster and a 40 m pixel edge.

169. Compare the plant cover maps resulted from tasks 164 and 167.

According to this tutorial, the names of these files should be: *5454_plant_cover_classes.rst* and *5454_plant_cover_classes_v3.rst*.

170. Compare the similarity maps that resulted from tasks 164, 165 and 167.

According to this tutorial, the names of these files should be: *5454_sim_to_predicted_class.rst*, *5454_sim_to_predicted_class_v2.rst* and *5454_plant_cover_classes_v3.rst*.

7.4. Substitute features

Constud uses features that are switched on in *[EXPL_VAR].[F...]* when calculating indices and learning recognition of dependent variables. Normally there is no need to replace or duplicate features. While mapping, a remarkable amount of time can be saved by mounting more complicated and repeatedly used features to be used as additional data layers. Every feature can be replaced by an additional data layer that is registered in the *D_LAYERS table*. **Constud**'s ability to calculate indices to raster is appropriate to exploit for creating data layers containing replacement features (Chapter 3.2).

The replacement features can be defined in the *EXPL_VAR* table as substitute features. In this case the same feature is essentially recorded twice in the *EXPL_VAR* (Fig 7-9) table. The substitute feature used in mapping must have a larger AID value than the feature used in learning. The automated replacement of features by their substitutes is applied only for mapping and not while learning or calculating spatial indices.

A substitute feature should be fast to compute. The best options in **Constud** are: index number 102 (the mean value) without sampling (sample *1* or *DB null*) read from the foal pixel (radius 0 or *DB null*) for a numerical feature; and for a nominal feature — index number 2, (the first order mode) analogously without sampling and read from the focal pixel. The time spent on calculating these features is negligible.

Another option to register a feature for which values are calculated to an additional layer is the direct replacement of feature attributes in the *EXPL_VAR table*. This means updating fields *KID*, *Index_ID* and *radius* in *EXPL_VAR*. The flaw of the direct replacement is that it destroys the data structure used in **Constud** learning. Before restarting learning, the changes in table *EXPL_VAR* must be reversed or the features in learning data altered in the same way as for prediction. The more modifications are made to the database, the higher the risk of producing errors in the complicated data structure necessary for **Constud** software.

EXPL_VAR

AID	name	KID	index_ID	radius	substitute
67	slope aspect	3	109	500	
68	slope aspect	50	102		67
69	weighted mode	1	11	1000	
70	weighted mode	51	1		69

Figure 7-9. A fragment of the *EXPL_VAR* table with features 67 and 69 used in **Constud** learning and their substitutes: AID = 68 and AID = 70. While mapping in **Constud**, the generalised slope aspect is not calculated from pixel values within 500 m in the data layer KID = 3 at every output pixel, instead the value of the focal pixel in data layer KID = 50 is used. Analogously, the distance weighted mode within 1000 m is not calculated but rather input from the substitute data layer KID = 51.

Questions

1. Does **Constud** use interpolation polygons if the initial distance of interpolation is 1?

2. What happens if a data layer needed for production of a predictive map is missing among **Constud** data layers?

3. Is it possible that a location is predicted to be a present site for more than one category of the same nominal variable?

4. Are the predictive maps formed as a result of tasks 164 and 167 different? If yes, then in what ways?

5. How many different plant cover categories are in the map produced according to task 164? What is limiting the number of categories in this map?Are the similarity maps produced according to tasks 164 and 167 different?

6. Are the maps depicting similarity to a given category and similarity to exemplars used different? If so, then which one contains higher similarities as a rule? Why?

7. Can a predictive map produced in **Constud** have irregular shape?

8. Why was the number of data layers in map sheet 1 used to map the mean amount of precipitation in the Baltic countries?

Summing-up

Constud offers more options than are reasonable to include in a manual meant for beginners. The discovery some merits and nuisances are left to the user to find. For example, the opportunity to use stored similarities between categories of a nominal variable, and recorded values of standard deviations of numerical features are only mentioned but not illustrated by practical exercises.

A software solution develops while it has users. The manuals and tutorials have to improve when a software system advances. Each suggestion and critical remark sent from http://digiarhiiv.ut.ee/Constud2/CS_Tutorial.aspx or via personal communication with the authors helps us along with the development of the system and the documents.

Constud software was largely modified while writing this tutorial. We made efforts to exclude errors but nobody is perfect. Modification in one part of a complicated system may cause undesirable effects in another module or while using already verified settings. The authors look for understanding from the users and support in developing **Constud**'s system.

References

Allouche, O., Tsoar, A., Kadmon, R. 2006. Assessing the accuracy of species distribution models: prevalence, kappa and the true skill statistic (TSS*). Journal of Applied Ecology*, 43, 1223–1232.

Congalton, R.G., Green, K. 1999. *Assessing the Accuracy of Remotely Sensed Data: Principles and Practices*. Lewis Publishers.

Remm, K. 2004. Case-based predictions for species and habitat mapping. *Ecological Modelling*, 177(3-4), 259–281.

Remm, K., Linder, M., Proosa, H. 2010. *Description of the Software System Constud 2. Version 27.08.10*. Chair of Geoinformatics and Cartography, University of Tartu (http://digiarhiiv.ut.ee/Constud2/Doc/Software_system_Constud.pdf).

Remm, K., Remm, L. 2009. Similarity-based large-scale distribution mapping of orchids. *Biodiversity and Conservation*, 18(6), 1629–1647.

Remm, K., Remm, M. 2010. Geographical aspects of enterobiasis in Estonia. *Health & Place*, 16, 291–300.

Remm, M. 2006. Distribution of enterobiasis among nursery school children in SE Estonia and of other helminthiases in Estonia. *Parasitololgy Research*, 99,729–736.

Remm, M., Remm, K. 2008. Case-based estimation of the risk of enterobiasis. *Artificial Intelligence in Medicine*, 43(3), 167–177.

Tamm, T., Remm, K. 2009. Estimating the parameters of forest inventory using machine learning and the reduction of remote sensing features. *International Journal of Applied Earth Observation and Geoinformation*, 11(4), 290–297.

Appendices

Appendix 1. Terms and abbreviations

ADSD — the archive of digital spatial data at the Institute of Ecology and Earth Sciences (description at http://digiarhiiv.ut.ee).

Attribute — a property of an object.

Continuous variable — a variable for which any numerical value within the range is possible. The alternatives are nominal variables and discrete variables.

Cross-validation — testing of estimations obtained by a model or other predictive system on an independent dataset.

Data mining — the process of extracting patterns from large datasets by combining methods from statistics and artificial intelligence with database management.

Database field — a place where you can store data; visible as a column in the tabular view.

Database object — logical units within a database to store or reference data. Tables, queries and views of a relational database in the context of this tutorial.

Database schema — a section of a database: a set of user rights and database objects.

Database view — a stored query accessible as a virtual table composed of the result set of a query.

Dependent variable — the predictable variable in mathematical models and case-based systems.

Deterministic process — a process in which no randomness is involved. A deterministic process will always produce the same output from a given initial state.

Discrete variable — variable that can assume only a finite number of values.

Divisor — an integer dividing values of a variable leaving a remainder. In **Constud**, any number dividing values of a feature.

Exemplar — a case or observation record selected to a predictive set for similarity-based reasoning.

Feature — an explanatory (independent) variable in machine learning systems.

Feature vector — an n-dimensional vector of numerical features that represent some object.

Goodness-of-fit — a summarizing statistics describing how well the estimates and observed values overlap.

Kernel — in **Constud,** a moving window, in which pixels are included in the calculation of a spatial index.

Learning data — a data set or sample used to calibrate the parameters of a model or other predictive system; also called training data.

Learning fit — adequacy of fit in the learning sample.

Learning iteration — one stage in the iterative optimization of the predictive set of features and exemplars. Learning iterations in **Constud** involve different techniques for optimizing weights for features and exemplars. These optimizations also proceed iteratively.

Leave-one-out cross-validation (LOOCV) — a validation method that involves using every single observation from the original sample as the validation data, and the remaining observations as the training data. Each observation in the sample is used once as the validation data.

Machine learning — a process during which an artificial system improves knowledge included to the system. Also a scientific discipline. In **Constud**, the iterative optimization of weights for features and observations. The weights are optimised to reduce prediction errors of similarity-based predictions in learning data. **Constud** has memory: the results of previous iterations affect the next iterations.

Mode — the most common value of a variable.

Multinomial variable — a nominal (categorical) variable which has more than two possible values.

Nominal variable (categorical variable) — a variable that consists of values describing the membership of cases to a particular category.

Number of degrees of freedom — the number of free components in the model. In general, the degree of freedom of an estimate is equal to the number of

independent observations minus the number of parameters in the predictive model.

Objective function — a function or statistic that has to be minimised or maximised while choosing the most correct model.

Parsing — a process of resolving a complex object into its elements that can be easily stored or manipulated.

Partial similarity — similarity between cases in the aspect of only one feature.

Pearson's coefficient of correlation — a measure of dependence between two variables obtained by dividing the covariance of the two variables by the product of their standard deviations.

Pre-classifier — a characteristic dividing data into subsets and enabling separate data handling in every subset.

Raster format — representation of spatial data as a regular grid: an array of pixels.

Root mean square error (RMSE) — a measure of the differences between predicted or estimated values of a numerical variable and the values actually observed; calculated as the squared root from the mean of squared biases.

Sample size — the number of observations in a sample.

Seamless spatial data — data structure where subset borders (map sheet boundaries) are not visible.

Single format — single-precision floating point computer numbering format, a binary that occupies 4 bytes (32 bits).

Spatial autocorrelation — distance-related dependency between values of the same variable. In case of positive spatial autocorrelation, close locations tend to be more similar than distant ones in regards to this variable.

Spatial pattern — arrangement of objects or values in space.

SRTM (Shuttle Radar Topographic Mission) — an international project headed by the US National Geospatial-Intelligence Agency (NGA) and the US National Aeronautics and Space Administration (NASA). The project obtained digital elevation data using a radar system that flew on-board the Space Shuttle Endeavour during an 11-day mission in February of 2000.

Validation fit — goodness of fit estimated in a validation sample.

Vector format — a data format that stores spatial data in vectors (attributes of objects and coordinates of nodes) rather than regular pixels, allowing for easier scaling and reduced storage.

Appendix 2. Data tables required for the English version of Constud

Appendix 2.1. Required fields in the *DEP_VAR table*

Field name	Format	Comment
calc	yes/no	Whether to calculate this variable or not
FID	byte	ID numbers of dependent variables
given	byte	The code of the category learned separately from other categories of the same nominal variable
logID	4-byte integer	ID of hitherto the best result corresponding to the matching row in the respective LOG table
precl	byte	ID of the feature used as a pre-classifier
name	text	A name given to the dependent variable by the user
ftype	byte	The type of the dependent variable
no_part_feature	byte	The number of dimensions (partial features) of a multidimensional numerical variable
sumsimmax	4-byte real	The initial value for the sum of similarity searched for decision making
dataquery	text	The name of the database object containing learning observations and fields for results
t_sample	4-byte integer	Approximate size of the learning sample (needed only for machine learning). Values >2000 are automatically changed to 2000
no_features	4-byte integer	Approximate number of features used in learning iterations
v_sample	4-byte integer	Approximate size of the validation sample (needed only for machine learning). Values >5000 are automatically changed to 5000
0_dist	4- byte real	A distance correction factor for reducing the effect of spatial autocorrelation (needed only for machine learning of spatial variables)

The combination of values in the fields *given* and *FID* must be unique for the rows where *calc* = *True*.

Empty field *precl* or values 0 or 255 mean learning and/or prediction without pre-classification.

The types of dependent variable specified in the field *ftype* are coded as follows: 0 = multinomial, 2 = numerical, 3 = binomial, 4 = one category of a multinomial variable opposed to all other categories of the same variable. Code 1 is reserved for multidimensional numerical variables, the option is hitherto not in functionality. If *ftype* = 4, a separate row for each class must exist in the *DEP_VAR* table.

The sum of similarity (*sumsimmax*) is the total value of similarity between the

predictable observation and the exemplars, after which the process of searching for the next similar exemplar is stopped.

The distance correction parameter (field *0_dist*) decreases similarity between observations proportionally to the inverse distance between observation sites, hindering reciprocal prediction between closely located observations. If the distance between sites is less than the value of *0_dist* the similarity between observations is turned to zero.

The size of sample is calculated combining random probabilities and the expected proportion of the sample; the actual sample size of learning iterations is stored in the corresponding log table (Appendix 2.4).

Appendix 2.2. Required fields in the *EXPL_VAR* table.

Field name	Format	Comment
AID	integer	ID of the explanatory features
name	text	Name of the feature given by the user
KID	byte	ID of the data layer in table D_LAYERS
index_ID	byte	ID of the spatial index in table SP_Indices
radius	4-byte real	Kernel radius (needed only for calculation of spatial indices)
in_radius	4-byte real	Internal radius of an annulus kernel (needed only for calculation of spatial indices)
sample	4-byte real	Approximate sample proportion in the range 0...1 (not required for machine learning)
blocksample	yes/no	The use of block-sampling (not required for machine learning)
given	byte	A given code is needed for indices expressing frequency of a category or distance to a category
precl	yes/no	The use of a pre-classifier to restrict the calculation of spatial indices to a certain category
precl_layer	byte	KID of a nominal data layer that is used as a pre-classifier. Empty field or 0 means the pre-classifier is not applied
BO_folder	text	Folder of the binary output if the index is calculated to a raster layer (needed only for the calculation of indices to a raster layer). Raster output is calculated from all files having extension rst in the folder of the data layer connected to the selected feature.
substitute	byte	ID of a feature replaced by this feature. Value of the field must be empty or 0 if the feature is not a substitute feature. Substitute features can be used only for spatial data.
divisor	single	Divisor used to divide the values of indices calculated from an integer format data layer.
F1, F2, F3 …	yes/no	The involvement of features into calculation of indices and/or machine learning. The field is relevant if the matching FID in table DEP_VAR is switched on. E.g. DEP_VAR.F1 directs Constud to EXPL_VAR.F1. Applied if ftype ≠ 4.
AF1, AF2, AF3, …	byte	Actuality of features. Applied if ftype ≠ 4.
F1_1, F1_2, F1_3, …	yes/no	The application of features for calculation of indices and in machine learning of a category of a nominal dependent variable FID = 1 having the marked given code. Applied if ftype = 4.

Spatial pattern indices are calculated within circular kernels having a user-defined radius. For features indicating distance to a particular category, calculations should be made without a pre-classifier and within a rather large radius. Feature values at the centre of the observation can be calculated using radius = 0.

Appendix 2.3. Required fields in learning data.

Field name	Format	Comment
VID	8-byte integer	Identifier of observations
F	4-byte real if ftype = 2, yes/no or byte if type =3, byte if ftype = 1 or 4	Value of the dependent variable
pred	same as for field F	Value predicted using learning data (needed only for machine learning and predictions stored in database table)
sim	byte	Similarity to exemplars chosen from the rest of learning data (needed only for machine learning and when predictions are stored in a database table)
actuality	byte	Actuality of the observation (needed only for machine learning)
w	4-byte real	Weight of observation as an exemplar
x	8-byte double	West-east direction in Cartesian coordinates
y	8-byte double	South-north direction in Cartesian coordinates
from	date, format depends on op-system settings	Beginning of the observation validity. Can be empty.
until	date, format depends on op-system settings	End of the observation validity. Can be empty.
<feature number>	byte	Feature values

Appendix 2.4. Required fields in log tables.

Field name	Format	Comment
ID	8-byte integer	ID of learning iteration
given	byte	Code of the category that is learned separately from the other categories of the same nominal variable
date	date	Date and time of saving the record
t_fit	4-byte real	Learning fit of observations and predictions
v_fit	4-byte real	Validation fit of observations and predictions
sumsimmax	4-byte real	The sum of similarity required for decision
etn	4-byte integer	The number of exemplars selected as a result of learning iteration
sample	4-byte integer	Learning sample size
F_weights	memo	Weights for features as: [no of a feature] [weight] [no of a feature] [weight], e.g.: 32 1.2342 35 0.328 87 0.9387.

Appendix 2.5. Required fields in the table *D_LAYERS.*

Field name	Format	Comment
KID	2-byte integer	ID of the data layer
p_edge	4-byte float	Pixel edge in the same units as coordinates of locations.
folder	text	Path to the files of this data layer without prefix input through the main dialog window
nominal	yes/no	yes means layer values will be treated as nominal
void	byte	Pixel value that is treated as absence of the data layer
from	date, format depends on op-system settings	Date of the data layer
until	date, format depends on op-system settings	Valid through date of the data layer
bytes	byte	Data format: 1 = byte, 2 = short integer; nominal layer can be in byte format only

Appendix 2.6. Fields in the *Map_sheets* table. First three fields are required for the calculation of pattern indices in case of square map sheets.

Field name	Format	Comment
NR	4-byte integer	ID of map sheet
min-x	8-byte double	Cartesian coordinate of the western border of the map sheet
min-y	8-byte double	Cartesian coordinate of the southern border of the map sheet
max-x	8-byte double	Cartesian coordinate of the eastern border of the map sheet (unnecessary if only square-shape map sheets are used)
max-y	8-byte double	Cartesian coordinate of the northern border of the map sheet (unnecessary if only square-shape map sheets are used)

Appendix 2.7. Required fields in self-similarity tables.

Field name	Format	Comment
1	byte	Category of the first nominal feature
2	byte	Category of the second nominal feature
s	byte	Similarity in scale 0…100 (percentage points)

Appendix 2.8. Required fields in the *SP_Indices* table.

Field name	Format	Comment
Index_ID	byte	ID of the index
name	text	Name of the index
nomlayer	yes/no	yes means the index is calculated from nominal layer
nomind	yes/no	yes means the values of the index represent nominal categories

Appendix 2.9. Attributes of pattern indices in **Constud**. The values of indices are handled as class codes of a nominal feature if nomlayer = *yes* and as numerical features if nomlayer = *no*. If nomind = *yes* then the index is nominal, if *no* then numerical.

Index_ID	name	nomlayer	nomind
1	share of given class	yes	no
2	mode	yes	yes
3	Shannon	yes	no
4	Lloyd	yes	no
5	dominance	yes	no
6	number of classes	yes	no
7	class adjacency	yes	no
8	direction of patches	yes	no
9	class proximity	yes	no
10	share of different class pairs	yes	no
11	distance weighted mode	yes	yes
12	distance to class boundary	yes	no
13	distance from a given class	yes	no
14	distance weighted frequency	yes	no
15	second mode	yes	yes
16	third mode	yes	yes
101	share of values above mean	no	no
102	mean	no	no
103	standard deviation	no	no
104	median	no	no
105	Moran's I of 8 neighbours	no	no
106	distance weighted Moran's I	no	no
107	mean difference of adjacent pixels	no	no
108	coefficient of variation	no	no
109	gradient direction	no	no
110	difference of border to centre	no	no
111	minimum	no	no
112	maximum	no	no
113	factor of kurtosis	no	no
114	gradient smoothness	no	no
115	difference from mean	no	no
116	gradient intensity	no	no
117	distance weighted mean	no	no
118	stripeness	no	no

Appendix 3. Spatial indices in Constud

3.1. Indices calculated from nominal data layers

Proportion of a given category — the share [%] of a given class within a given kernel.

Mode — the code of the category having the largest share in the kernel.

Shannon's index of diversity — the diversity of pixel classes according to the formula

$H = -10\sum pi \log_2 p_i$, where p_i — share of the class i in the kernel.

Lloyd's index of equitability — calculated according to the formula $10\ H / lg(s)$, where

H — Shannon's diversity, s — number of classes within the kernel.

Dominance — calculated using the formula $100 \sum p_i^2$, where p_i — share of the class i in the kernel.

Number of classes — the number of different categories in the kernel.

Class adjacency — the ratio of edges between the pixels of the same class to the number of all edges. Equals 100 when all pixels within the kernel belong to the same category.

Direction of patches — 0 means one-pixel-wide vertical stripes, 90 means horizontal one-pixel-wide stripes. A value of 255 marks the absence of patch borders.

Class proximity — the ratio of the sum of inverse distances between pixel centres of the same class to the sum of inverse distances of all pixel pairs; presented in percentages.

Share of different class pairs — the ratio of pixel pairs with pixels belonging to different classes to the number of all pixel pairs within the kernel; presented in percentages.

Distance weighted mode — the mode weighted by inverse distance. The category for which the sum of inverse distances of pixels from the kernel centre is the greatest. The focal pixel of the kernel is not included.

Distance to class boundary — the minimum distance in pixels from the kernel centre to the closest class boundary in pixels of the input layer.

Minimum distance from a given class — the distance measured in pixels from focal pixel to the closest pixel of a given category.

Distance weighted frequency — frequency of a given class weighted by inverse distance.

Second mode — the next frequent category after mode.

Third mode — the third frequent category after mode and second mode.

3.2. Indices calculated from numerical data layers

Share of pixel values above the mean — the share of pixel values [%] exceeding the mean value within the kernel expressing the asymmetry of the distribution of values.

Mean — the arithmetic mean of pixel values within the kernel.

Standard deviation — the square root of the sum of squared deviations of pixel values from the local mean.

Median — a pixel value for which the number of smaller pixel values is equal to the number of pixels having higher values.

Moran's I of 8 neighbouring pixels — spatial autocorrelation according to Moran's *I* of 8 neighbouring pixels. 0 means a maximum possible negative spatial autocorrelation, 100 means absence of spatial autocorrelation and 200 the maximum positive spatial autocorrelation (similar values adjoin).

Distance weighted Moran's I — the inverse distance weighted Moran's *I* having the same scale as the previous index.

Difference of neighbouring pixels — the mean difference between adjacent pixels.

Coefficient of variation — the ratio of the standard deviation to the mean × 100.

Gradient direction — direction of inclination of the linear trend surface within the kernel. 255 means no gradient, 50 — increasing values in direction of y-axis,

150 — decreasing values in direction of y-axis, 100 — increasing values in direction of x-axis, 1 and 200 — decreasing values in the direction of x-axis.

Difference of border to centre — difference between the mean value of pixels within half radius and the mean value of pixels located between half and full radius of the kernel. A constant = 100 is added to the difference and values less than 0 and above 254 are truncated, so the final statistic is in the range of 0–254, 100 meaning no difference between the central and peripheral part of a pixel.

Minimum — the minimum value within a given kernel.

Maximum — the maximum value within a given kernel.

Factor of kurtosis — calculated using the formula:

$$E = \frac{10}{n}\sum\left(\frac{x_i - \bar{x}}{\sigma}\right)^4$$

where x_i — pixel value, $\bar{x}$ — the mean of pixel values, n — the number of pixels within the kernel and σ — the standard deviation of pixel values. Greater values appear where areas of different brightness are close together.

Gradient smoothness — homogeneity of neighbouring pixels. Calculated using the formula:

$$\mathrm{S} = 100 \cdot \frac{\sum_{\mathrm{i}}\sum_{\mathrm{j}} \frac{1}{1+\left(x_i - x_j\right)^2}}{n},$$

where x_i and x_j — pixel values of neighbouring pixels in horizontal, vertical and diagonal directions and n — the number of pixels within the kernel.

Difference from the mean — the difference between the focal pixel value and the arithmetic mean of all pixel values within the kernel.

Gradient intensity — the angle of inclination of the linear trend surface of pixel values within the kernel. Difference of pixel values within the kernel to the mean value is multiplied by pixel distance in the direction of the gradient. Values range from 0 to 200. Zero means no gradient (the sum of weighted pixel distances on the gradient equals the sum of weighted pixel values on the gradient). Value 10 means the pixel values on a gradient change by one when the distance is increased by one pixel.

Distance weighted mean — the mean of pixel values weighted by inverse distance from the focal pixel.

Directional structures — stripeness calculated first as the largest difference between the average of 9 pixels and 3 pixels in a line in four directions (north to south, east to west, northwest to southeast, and northeast to southwest) around every pixel and thereafter as the mean of all partial differences within the sample in the direction of the maximum difference. The stripeness = 0 if no directional structures are present.

Appendix 4. Answers and comments to questions

1-1. Yes — after vector polygons are rasterised.

1-2. Yes, if a rasterised map of running waters exists.

2-1. **Oracle** connection provider is not yet included in **Constud** software.

2-2. **MySQL** connection provider is not yet included in **Constud** software.

2-3. 35.

2-4. Indices that are calculated from nominal layers have ID numbers less than 100; indices calculated from numerical layers have ID >100.

2-5. 2, 11, 15, 16, mode, distance weighted mode, second mode, third mode.

2-6. *[DEP_VAR].[dataquery]*.

2-7. Equal to the number of nominal variables.

2-8. *[DEP_VAR].[p_edge]*, *[EXPL_VAR].[radius]*, *[EXPL_VAR].[in_radius]*, and also *[X]* and *[Y]* in tables of observations, predicted values, *[min-x]*, *[min-y]*, *[max-x]* and *[max-y]* in the *Map_sheets* table.

2-9. In the field *[DEP_VAR].[log_ID]*.

2-10. Pixel edge in units used in this **Constud** database.

2-11. Yes.

2-12. Yes, if the user accepts that degrees are treated as equal rectangular distances.

3-1. Negative land forms occupy a larger area.

3-2. The answer should be an xy-diagram.

3-3. 574. It is easy to reach the answer using the following SQL query:

```
SELECT Count(VID) FROM OBSERVATIONS1 WHERE [6]<>[7];
```

3-4. 4.120 and 1.029. Pixel values are more variable around open locations. It is easy to find the answer using the following SQL query:

```
SELECT StDev([6]) FROM OBSERVATIONS1 WHERE [1]=0;
SELECT StDev([6]) FROM OBSERVATIONS1 WHERE [1]=100;
```

3-5. 8 locations. You can reach the answer using SQL query:

```
SELECT Count(VID) FROM OBSERVATIONS1 WHERE [9]>[8];
```

3-6. 255, soil types are not mapped for lake areas.

3-7. The answer depends on the results of task 40.

4-1. 1 and 0.

4-2. ftype = 4, which is separate learning of categories of a nominal variable.

4-3. Because the weights are fitted to the learning sample. The opposite is also theoretically possible.

4-4. No, weights and actualities are not the same.

4-5. Actualities are a result of learning and are used in the following learning iterations. Weights of features and exemplars are not used in the following learning iterations; they do not pass information from a learning iteration to the next ones.

4-6. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

4-7. These are the only representatives in their pre-classifier unit. If a pre-classifier is included, exemplars can be used only within the same pre-class. Prediction by itself is not allowed in **Constud** learning.

4-8. The amount of precipitation is the smallest at Sõrve meteorological station. The predicted values depend on results of **Constud** learning, which is not deterministic but a semi-random process. The minimum is in most cases not predicted to Sõrve because prediction by itself is not allowed in **Constud** learning.

4-9. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

5-1. No. The browsing window uses a local copy of the original table.

5-2. The answer depends on the results of tasks 90–93.

5-3. The answer depends on the results of task 99.

5-4. No.

5-5. No.

5-6. If samples were used in learning.

5-7. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

5-8. A pre-classifier was used in plant cover learning; each pre-class has separate exemplars.

5-9. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

6-1. 0-1.

6-2. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process. In most cases, the result is that the third child in the table is enterobiasis positive, and the others are negative.

6-3. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

6-4. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

6-5. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process. The following queries will give the answer:

```
SELECT Avg(sim) FROM Plant_cover_test WHERE [F]=[pred];
SELECT Avg(sim) FROM Plant_cover_test WHERE [F]<>[ pred] and
[pred]<>255;
```

6-6. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process.

6-7. The observed amount of precipitation is more variable because **Constud** calculates the predicted value of a numerical variable as the weighted average.

6-8. The answer depends on results of **Constud** learning, which is not deterministic but a semi-random process. In most cases, the answer is field.

6-9. The predictable sites were not involved in learning.

7-1. Is used only to define excluded area.

7-2. **Constud** shows a warning message and calculates using the rest of the features.

7-3. Yes, if predictions are calculated separately for single categories of a nominal variable.

7-4. These maps should not differ.

7-5. The number of units in this kind of predictive map is limited by the categories that are switched on in the field *[DEP_VAR].[calc]*.

7-6. These maps should not differ.

7-7. In general, these maps are different. Similarity to exemplars used in prediction is mostly higher than similarity to exemplars of a given category because the given category is not always the most similar.

7-8. The output raster is always rectangular. The mapped area within this rectangle can be irregular.

7-9. Because the same number is used in the names of used data layers.